Lucene for n-grams using the ClueWeb collection

Gregory B. Newby, Chris Fallen and Kylie McCormick
Arctic Region Supercomputing Center
University of Alaska Fairbanks, Fairbanks, AK 99775

Abstract

The ARSC team made modifications to the Apache Lucene engine to accommodate "go words," taken from the Google Gigaword vocabulary of n-grams. Indexing the Category "B" subset of the ClueWeb collection was accomplished by a divide and conquer method, working across the separate ClueWeb subsets for 1, 2 and 3-grams.

1. Overview and Prior work

Phrase searching—or imposing an order on query terms—has traditionally been an expensive IR task. One approach is to use sophisticated algorithms at query time to analyze the sequence of a given term relative to nearby terms in the text, where term locations were stored at indexing time. Another method is to index the document collection relative to a large set of phrase tokens, rather than single terms. For the TREC 2009 Web track, we indexed the ClueWeb09 Category B document collection utilizing a "go list" vocabulary (as opposed to a "stop list") of 1-, 2-, and 3-gram phrase tokens extracted from the Google *N-Gram* data set. We made small modifications to Lucene to facilitate use of the go list, in both the indexer and the query processor. We found that query processing was quite fast, with queries being processed in well under 2 seconds each. Lucene indexing time increased approximately 3X for the three passes (with 1-, 2-, 3-grams), and index files were larger due to the larger number of indexed n-grams. The additional cost paid to index using n-gram tokens allowed for phrase searching to occur as quickly as regular single term searching.

Prior TREC systems that we built for the Million Query (MQ) [2, 5] and Terabyte (TB) [4, 3] tracks were models of large-scale distributed or grid information retrieval (IR) systems, made from the GOV2 corpus distributed over various clusters from the Arctic Region Supercomputing Center (ARSC). Effective and efficient distributed IR is more difficult than monolithic IR, but it allows for easier fine-grained access control of document collections while maintaining search and discover capability.

Incorporating phrase search in distributed IR systems is useful, not just for the traditional retrieval task performed by each search node, but potentially also for intelligently restricting the number of collections searched. The search portal component of our 2008 MQ track system [5], for example, used a set of term-document co-occurrence matrices constructed from approximately 1000 document collections and a fixed 400,000 term vocabulary from the SCOWL wordlist [1], in order to restrict the number of collections queried using an eigensystem approach adapted from Latent

Semantic Indexing (LSI) techniques. Incorporating phrases into the master vocabulary of the portal collection-selection algorithm is a natural extension of that technique.

Our 2009 Web Track system provides search for a set of document collections where each collection has been indexed relative to a fixed phrase vocabulary that has been previously constructed and filtered. The vocabulary may, or may not, be shared with multi-search portals that could use this information to rank or otherwise restrict either the set of collections to be searched or the set of documents to be retrieved from each remote collection. As with our previous TREC systems, the ARSC Web Track phrase-search application described here is built on a modified version of the Lucene Toolkit. Thus, similar phrase-search functionality could be incorporated with existing and well-tested search APIs.

2. Component Description

In this paper, we define a *token* to be any number of one or more characters, numbers, or punctuations (including whitespace) accepted as a single 'word' in the vocabulary. A *term* refers here to any number of one or more characters, numbers, or punctuations (excluding whitespace) within a token that is separated by white space. In other words, an n-gram token has *n* terms.

Lucene

Lucene is an open-source tool from the Apache project that contains a high-performance search engine library written in Java [6]. Lucene has a simple API with complex options for building and maintaining search engines with scalable, high-performance indexing and searching. Lucene only processes plain text documents, so HTML and other markup languages must be removed prior to indexing.

Lucene discovers document tokens by using a *TokenStream* from each document. Each token is defined by the chosen *Tokenizer* object, which populates the TokenStream by breaking up the full plain text input of the document. *TokenFilters* are used to modify, remove, or even add tokens to the TokenStream.

The Google Gigaword collection (described below) provided us with a “go-list” of word tokens that we wanted to index, rather than a “stop-list” of word tokens that we did not want to index. We used the Lucene *WhitespaceTokenizer* to populate the TokenStream from any given document, and then applied our *GoWordFilter* that ignored all unrecognized terms.

The first index (see section 3) we built for the Web track was used during out official Web track run and utilized only unigram tokens. We also indexed the data using phrases, or n-grams, from the Google Gigaword collection, and this work was used for subsequent non-official runs. For the n-gram index and search task, we use Sebastian Kirsch’s *NGramFilter* for Lucene to populate the TokenStream with all possible n-grams from the document before applying the GoWordFilter, which ignores any unknown n-grams in the TokenStream.

Google Gigaword

The Google Gigaword collection is comprised of unigram, bigram, trigram, fourgram, and fivegram tokens found and normalized from approximately one trillion word tokens extracted from

publically accessible web pages[1]. While only English-text web pages were candidates, some non-English texts and words exist in the Gigaword collection. The collection also included the token occurrence frequencies across these documents. The tokens were normalized by length and character formatting as well as occurrence value. If a unigram token appeared at least 200 times, or an n-gram token appeared at least 40 times, the token was kept in the Gigaword collection.

This collection also maintained cases; so "the," "The," and "THE" were all considered separate unigrams. The same goes for punctuation; there is a distinction between the tokens "word!" and "word." These two particular distinctions were not relevant to our indexing, so we preprocessed the token files by removing tokens considered redundant. Tokens that had unrecognized characters or were comprised of only punctuation marks were removed. However, tokens that contained punctuation marks were retained, except in the case of an end mark (?, !, ., ;, ,) at the end of a word or an unpaired parenthesis.

The final alteration of the tokens in the Gigaword set was collapsing all like Date Tokens. Since dates can be represented in many forms but be the same date (i.e. 15 May 2009 and 05/15/09), a simple date filter was used to identify possible date tokens. If the token parsed into a date (containing at least a month and either a year or a day, but possibly all three), it was changed into unified date token (formatted: YYYY-MM-DD or YYYY-MM or MM-DD). The date filter works on both unigrams and n-grams, and it can modify an n-gram partially if only part of the n-gram contains a date value (i.e. Superbowl Sunday Feb. 1, 2009 would become Superbowl Sunday 2009-02-01).

The preprocessing (dubbed "cleaning") of the Category "B" ClueWeb data took approximately 1705 CPU hours to compute. This computation also provided files that did not contain the occurrence values of each token, which is reflected in the file sizes reported in Table 1. The large difference between the fourgram and the fivegram collection sizes may be attributed to the fact that any rejected term found within an n-gram token would cause the entire token to be discarded. The same factors that would discard a unigram, described above, would cause a term to be discarded, except that a term within an n-gram token could be a length of one character; whereas, unigram tokens needed to have at least two characters.

	Original Count	Original Size (GB)	Modified Count	Modified Size (GB)
Unigrams	13,588,391	0.17	11,041,700	0.099
Bigrams	314,843,401	2.5	214,323,478	2.6
Trigrams	977,069,902	9.8	465,538,519	8.5
Fourgrams	1,313,818,354	18	500,249,203	12
Fivegrams	1,176,470,663	11	168,251,989	4.8
Total	3,795,790,711	41	1,359,404,889	29

Table 1 Google Gigaword Tokens and file sizes before and after cleaning.

[1] This collection is available from the Linguistic Data Consortium as "Web 1T 5-gram version 1" at www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13

ARSC's *Midnight* Supercomputer

We utilized several computational systems to accomplish the work described here. For processing the ClueWeb dataset and running Lucene with the Google Gigaword "go-list," the primary system used was ARSC's "Midnight."

Midnight is a supercomputer cluster built from Sun Microsystems servers. With 2312 processors and a theoretical peak performance of 12.8 teraFLOPs (TF), Midnight consists of 358 SunFire x2200M2 "small" nodes (each with two dual-core Opteron processors) and 55 SunFire x4600 "large" nodes (each with eight dual-core Opteron processors). Compute nodes provide 4GB of memory per processor core. Connected via InfiniBand, programs on Midnight have rapid access to 140 raw TB of high performance storage on a Lustre filesystem. Midnight runs the SuSE Linux operating system.

We ran most jobs on the "large" nodes of Midnight, each with 64GB of memory. They are connected to a high-performance shared workspace, have 250GB of local non-shared temporary storage, and access to petabyte-scale archival storage. For the most part, we used the high-performance storage to store the ClueWeb collection and other input data, then the local on-node disk to build incremental indexes. Indexes were then copied out to the high-performance storage at the end of each indexing job. Multiple jobs, each with multiple Lucene threads, were needed to complete our indexing.

3. Indexing procedure

Before the ClueWeb data collection was indexed, we pre-processed the WARC files from the collection into plain text documents associated with the corresponding TREC IDs. This pre-processing stripped HTML and markup tags from each document, leaving behind meta-data (such as *alt* values from *image* tags) and some scripting values from languages like Javascript. Since the Gigaword collection contained primarily English words, we only indexed the English data from ClueWeb09. As Table 2 shows, the reduced plain text files were approximately 21% the size of the uncompressed, WARC-formatted files with markup tags.

	Compressed WARC Files (TB)	Uncompressed WARC Files (TB)	Plain Text Reduced Files (TB)
Class B Dataset	0.22	1.5	0.31
Full English Dataset	2.0	13.2	2.8

Table 2 ClueWeb09 English Dataset WARC file sizes.

The size of the data set made even a single, monolithic indexing sweep undesirable. Lucene offers index merging, which takes multiple indexes and merges them into one, which can then be optimized for searching. We used this feature both on the small scale and the large scale. First, on the small scale, we implemented multithreading indexers to create a set of small indexes built in parallel during the program's run; then, the final set of the programs merged them into a single, final index for this run. On the large scale, we ran this program once per every 70 – 100 reduced

files, depending on the exact file size. When all reduced files were indexed, we merged the final indexes of these program runs to make a single index.

The other benefit of merging was flexibility with the choice of vocabulary. We were able to index the reduced documents first with only unigrams, then with only bigrams, and then finally merge the two super-indexes together. This method was used for all sizes of grams to reduce the amount of memory used by Lucene, which enabled us to have more threads and a shorter token-load time, which sped up the process of indexing.

This divide and conquer scheme also enabled us to retain the smaller, sub-indices from each program run, while providing us with a stable monolithic index to search. These sub-indices can be used for later research, possibly in relation to our previous work in Multisearch.

4. Run description

The official ARSC09 submitted run used unigrams over only a fraction—about 25%—of the Class B data set. We subsequently indexed unigrams up to trigrams (with fourgrams and fivegrams pending), for later comparison. The searching was exceptionally fast, with only one query taking long enough—one second—for our timer to measure it. Table 3 gives a list of select queries, the number of found documents in the Class B data set and the time it took to run the query.

Query Name	Query Value	Found Documents	Time to Run Query (sec)
wt09-1	obama family tree	473,746	0
wt09-2	french lick resort and casino	2,129,269	0
wt09-3	getting organized	138,777	0
wt09-4	toilet	7383	0
wt09-40	michworks	0	0
wt09-47	indexed annuity	4948	1
wt09-48	wilson antenna	85,021	0
wt09-49	flame designs	60,759	0
wt09-50	dog heat	94,618	0

Table 3 Selected queries and their results and runtimes over the Class B data set.

Unigram runs simply search the index with the modified query terms, which undergo the same modifications as the indexed data, as described in section 3. Unigrams were run over both the Class B dataset and later over the full English data set of ClueWeb.

N-gram runs had two run types: exact and verbose. Exact n-gram runs modified the query such that the largest n-gram token in the query would be considered a single token. For example, the query *little red riding hood cookies* would recognize only two tokens: *little red riding hood* and *cookies*. Essentially, this is exact phrase matching.

Verbose n-gram runs modified the query by populating it with all valid renditions of its terms as n-gram tokens and removing any tokens that are not listed in the modified Gigaword vocabulary. The

original query terms are also preserved. For example, the query *little red riding hood cookies* would become the search tokens *little red*, *little red riding hood*, *hood cookies*, *little*, *red*, *riding*, *hood*, and *cookies*. Essentially, this is query expansion.

5. Preliminary results

Initial IR performance summary results from the ARSC09 Web track system were based only on the subset of Category "B" that was indexed in time for the submission deadline. Relative to both the estimated Average Precision (AP) and Normalized Discounted Cumulative Gain (nDCG) [7] performance measures, our Web track system scored below the TREC median on 90% of the topics and tied the TREC worst score on 60% of the topics. Further analysis is needed to identify system deficiencies; however, we indexed only 25% of the Category B data set before submitting the official results and this is likely a significant contributing factor in our sub-median IR performance. A baseline run with a stock Lucene-based system would also help to determine whether the phrase index and search technique improved or degraded Lucene IR performance.

Four topics—numbers 4, 23, 28, and 29—are especially interesting because they scored above the median on one or both IR performance measures (see Table 4). Our nDCG score on topic 29, *ps 2 games*, was the best of the TREC participants. Topics 4 and 28, *toilet* and *inuyasha*, respectively, scored above the TREC median relative to nDCG but tied the TREC worst relative to AP. Likewise, topic 23, *yahoo*, scored above the TREC median relative to AP but tied the TREC worst relative to nDCG. It is interesting to note that only one of the exceptional topics is a phrase token containing more than a single term. In particular, the topic *ps 2 games* likely performed well because each of the terms was contained in the vocabulary yet the number *2* was probably discarded by many other TREC participant systems during indexing.

Query	AP				nDCG			
	ARSC	TREC best	TREC median	TREC worst	ARSC	TREC best	TREC median	TREC worst
wt09-04: toilet	0.0141	0.2177	0.0902	0.0005	0.1170	0.4098	0.1082	0.0000
wt09-23: yahoo	0.0292	0.5492	0.0106	0.0000	0.0000	0.3271	0.0493	0.0000
wt09-28: inuyasha	0.0081	0.7384	0.3735	0.0081	0.3370	0.6450	0.3175	0.0422
wt09-29: ps 2 games	0.0081	0.0918	0.0065	0.0001	0.1782	0.1782	0.0000	0.0000

Table 4 IR performance of the ARSC Web Track system on topics scoring above the TREC participant median relative to the AP or nDCG performance measures.

6. Summary and future work

ARSC is continuing with several different approaches to experimentation in information retrieval. Along with other TREC Web track participants, we will be looking into characteristics of the ClueWeb collection, and seeking to understand performance on topics. Our research with a "go-word" list, described here, has promise for more easily enabling indexing and searching on phrases. We anticipate that, with further tuning, this will be a major advantage for TREC's Web-style queries. Such queries only have a few words, and therefore we can match the entire query as a phrase. To date, we believe Lucene has not yet been modified to give differential weights to phrases (versus

terms). Modifying TF/IDF or pivoted weighting schemes to account for phrases, versus single terms, is a goal.

We are also continuing with our investigations into large-scale eigensystems [8], and included some detail on work to date in our TREC 2009 poster. Using the PETSc and SLEPc packages, we have performed full eigensystems analysis (singular value decomposition) on sparse matrices including over 600,000 terms [9]. Description of the properties of these eigensystems, and their utility for information retrieval, will be the focus of a forthcoming paper.

Research described at past years of TREC is also ongoing. This includes our efforts on grid computing, federated search, and unified relevance ranking from discrete collections.

7. References

[1] K. Atkinson, *Kevin's Word List Page*, 2007.

[2] C. T. Fallen and G. B. Newby, *Collection Selection Based on Historical Performance for Efficient Processing*, *16th Text REtrieval Conference*, NIST Special Publications, Gaithersburg, Maryland, USA, 2007.

[3] C. T. Fallen and G. B. Newby, *Distributed Web Search Efficiency by Truncating Results*, *JCDL '07*, ACM, Vancouver, British Columbia, Canada, 2007.

[4] C. T. Fallen and G. B. Newby, *Partitioning the Gov2 Corpus by Internet Domain Name: A Result-set Merging Experiment*, *The 15th Text REtrieval Converence*, NIST Special Publications, Gaithersburg, Maryland, USA, 2006.

[5] C. T. Fallen, G. B. Newby and K. McCormick, *Distributed multisearch and resource selction for the TREC Million Query Track*, *17th Text REtrieval Conference*, NIST Special Publications, Gaithersburg, Maryland, USA, 2008.

[6] O. Gospodnic and E. Hatcher, *Lucene in Action*, Manning Publications, 2004.

[7] K. Järvelin and J. Kekäläinen, *Cumulated gain-based evaluation of IR techniques*, ACM Trans. Inf. Syst., 20 (2002), pp. 422-446.

[8] H. Kettani and G. B. Newby, *Instability of Relevance-Ranked Results Using Latent Semantic Indexing for Web Search*, *HICCS-43*, Honolulu, Hawaii, 2009.

[9] G. B. Newby, J. M. Styers and C. T. Fallen, *Large Scale Sparse Matrix Analysis for Term-Document Occurrences in Web Text*, *2009 Dynamics of Complex Systems*, Fairbanks, Alaska, 2009.

Acknowledgements

This work was supported in part by a grant of HPC resources from the Arctic Region Supercomputing Center and the DoD High Performance Computing Modernization Program.