Fujitsu Laboratories TREC7 Report

Isao Namba   Nobuyuki Igata   Hisayuki Horai   Kiyoshi Nitta
Kunio Matsui

Multimedia Laboratory Fujitsu Laboratories Ltd.
{namba,igata,horai,kiyoshi,kunio}@flab.fujitsu.co.jp

1 Abstract

In our first participation in TREC, our focus was on improving the basic ranking systems and applying text clustering techniques for query expansion.

We tested a variety of techiniques including reference measures, passage retrieval, and data fusion for the basic ranking systems. Some techiniques were used in the official run, others were not used because of time limitations.

We applied the text clustering techiniques for query expansion with a text clustering engine. Clustering base query expansion uses the top N best text clusters from the top 1000 documents instead of just using the top N documents.

Clustering base query expansion produces better results than simple query expansion based on passage retrieval.

We submitted three runs, Flab7at , Flab7ad, and Flab7atE. Flab7at is combination of ranking and query expansion by clustering the top 1000 documents on the title field, Flab7ad is combination of ranking and query expansion by clustering on the description field, and Flab7atE is combination of ranking with Boolean (existence) operators and query expansion by passage retrieval.

2 System Description

2.1 Overall

We particpated in TREC with two groups. One group was concerned with search engines, and included index construction, searching process, and normal query expansion. The other group was concerned with the Environment for Document Analysis (EDA), which is related to query expansion with text clustering.

The two groups had different locations, and the two systems were developped in completely different environments. To combine the two systems, we constructed an experimental search procedure using perl script. We also wrote a TREC local procedure in perl script for such tasks as removing stop patterns from input query.

2.2 The Search System Teraß

Teraß[1],[2] is fulltext search system, designed to provide an adequate number of efficient functions for commercial service, and to provide parameter combination testing and easy extension for experiment of information retrieval.

To satisfy both the commercial and experimental requirements, Teraß has many functions and extensibility as described below.

1. Basic search operations

   Boolean, Boolean Ranking (Ranking with Boolean operators for commercial use), Ranking (Accumulator Method) Near operators for phrase, Not operator, existing operator for term in ranking and parameter control, such as df limiter for ranking, term evaluation order control for ranking, etc.

2. Index type

   Inverted file index for fulltext search, number array index for range search, number array index for multiple occurences of number in a single text (eg. IPC code in a patent text) , and B-Tree index for item search.

3. Coding system for inverted file index compression

8-bit block coding (commercial standard)[3] δ, γ coding (academic standard)[4],[5], Extended γ coding [2].

4. Inverted file index construction

   Combination of term number, term frequency, and term offset in document

   With skip list[6],[4], and without skip list selection.

5. Ranking measures and easy extension to other measures

   Teraß supports reference measures such as Okapi, Lt.Lnu, and Cosine Measure, and is easily extended to other measures.

6. Other extensions

   (a) Parallel Processing[7], [8]
   Teraß can be extended to access multi indexes (data parallel) for large-size text collections or for getting better throughput on the AP-3000 parallel processor.

   (b) Language support and coding system
   Because the language-dependent part is separate, Teraß can be easily adapted to any language if the user writes a token analyzer. The current version supports English (stemming, soundex), Japanese (morphological analysis[9], Character N-gram), and Chinese (Character N-gram). Teraß also supports a variety of coding systems such as Unicode, Ascii, ISO8859, Shift-JIS, and EUC.

2.3 Environment for Document Analysis (EDA)

Environment for Document Analysis (EDA) assists in the analysis of many kinds of large document sets. It consists of several component tools. The major tool is Keyword Associator (KA) which provides the basic functions for full-text search engines based on vector space model, although it had originally been developed to support idea creation by presenting words related to a keyword[10]. The other components of EDA are KAcluster (part of KA), which provides a clustering function, Document Selector, which selects documents according to document attributes, and miscellaneous scripts and makefiles that combine EDA tools.

1. Keyword Associator

   Keyword Associator (KA) manipulates dictionaries of pre-calculated relevances between terms and documents. KA has five independent major parameters that determine the system behaviors of the basic functions: 1) Relevance pattern (RP), 2) Result type (RT), 3) Weighting measure, 4) Similarity measure, and 5) Normalization. Each of these parameters is described below.

   The functions provided by KA can be divided into four patterns:

   (RP-1) retrieve document from document

   (RP-2) retrieve term from document

   (RP-3) retrieve document from term

   (RP-4) retrieve term from term

   Each function pattern produces a result with three types of relevance:

   (RT-S) scalar value

   (RT-V) vector value

   (RT-M) matrix value

   Thus, 12 combinations of the RP and RT functions are available for obtaining relevances. The behavior of KA can be controlled by either command line options or by a KA-batch script language. KA-batch is an effective tool for document analysys and for reusing analysis techniques.

   The main dictionary of term-document relevance is produced from a set of documents. One of several term weighting measures, such as Lt.Lnu[11], tf, or idf, is applied to produce the dictionary. The relevance values of the retrieved results are calculated from the dictionary, using one of four similarity measures, Inner product, Dice coefficient, Cosine coefficient, or Jaccard coefficient[12]. The relevance is normalized either before or after the calculation. One of several normalization schemes (not apply, term vector sum = 1, term vector square sum = 1, document vector sum = 1, and so on) is applied.

2. KAcluster

   KAcluster reads the data of either the (RP-1, RT-M) or (RP-4, RT-M) types from KA and forms clusters of the resulting element. One of the hierarchical agglomerative clustering methods (HACM), such as Single link, Complete link, Group average, Median, Centroid, or Ward's method[13], is applied to perform the task. Usually, clusters generated by a clustering method are disjunctive partitions of the document set by determining an appropriate threshold. However, finding the threshold is known to be of the most difficult problems. Therefore, KAcluster generates all possible clusters produced from a hierarchical structure that the clustering method creates. The resulting clusters are not disjunctive partitions.

3. Document Selector

   Document Selector assigns an order relation to the given document set and selects documents from it. The document set may or may not be clustered. One or more combinations of ordering algorithms using the attributes of each of the documents is applied. For example, the attributes are the rank and score of the pilot search, and the attributes of the cluster the document belongs to, if the document set was clustered. The attributes of clusters are the average and median rank/score, the best and worst rank/score, and size.

3 Processing

The section introduces the overall TREC7 processing.

3.1 Preprocessing

1. Indexing

   The indexing vocabulary consists of character strings made up of letters, numbers, and symbols. For every string of alphabetic characters, stemming algorithm of Porter[14] was applied, and no stop words were used in indexing. Thus, all the tokens in the text are indexed including ",", ".", and "?" . This is because we are not certain as to what is reliable stop word list, how it affects the result. Also, we had not enough time to compare the different indexes.

   In the search, we used two adjacent terms within a specified distance as a phrase instead of using two adjacent words with a specified document frequency.

2. List of stop words for query processing

   We used stop word list of about 400 words of Fox[15], and added some missing words such as "near", "taken", "taking", etc..

   At run time, words with a high frequency (over 50000-80000) were also treated as stop words.

3. List of Stop patterns

   A TREC query has many peculiar expressions such as "find document which," "relevant document will describe," which are artificial, and which cannot be eliminated by a stop word list. Ideally, stop patterns should be collected from the input collection of actual users queries, or from a semantic analysis of the input queries, but we don't have them. So to remove them, we made an N-gram ($N = 1-4$) of the tokens from all of the TREC1-6 topics, and manually selected the stop patterns.

4. Synonym dictionary

   We made a simple synonym dictionary of words which appear frequently in TREC topics such as U.S., U.S.A, Japan, British, and U.N .

5. KA dictionary

   All documents were pre-processed by KA , then Lt.Lnu was used to generate a dictionary of relevance between the terms and the documents for query expansion by document clustering. This process took about ten hours on an AP-3000.

3.2 Query Processing

This section contains an outline of the query processing. We used different measures for the pilot search and for the final search after query expansion. The pilot search measure is designed for high precision, where as the secondary search measure is a reference measure in TREC (Okapi). For query expansion, we used two different methods, one method simply used the top N ranked documents, while the other method used the top N ranked clusters from the top 1000 documents.

1. Query generation (Teraß format query generation)

   (a) Stop pattern/word removal

   Remove the stop pattern from the topic and replace it with "*".

   (b) Stop word removal

   Remove the stop word from the topic.

   (c) Query generation

   Generate a query based on words, and phrases.

   A phrase is an adjacent pair of words that are not stop words, and is expressed as a two-word pair within a specified distance without order.

   The distance is set at 4 or 5.

   (d) Must operator attachment (optional)

   Attach the term existence operator by simple sentence pattern analysis.

   This operator means that the term must be in the search result.

   (e) Synonym dictionary expansion(optional)

   Expand synonyms such as U.S etc.

2. Pilot search

   Simple term frequency/text size with range limiting, and average size considering text size normalization is used , which seems to be ok in our experiment.

3. Query expansion

   Query expansion is based on a single term because we have found that automatically generated term pairs at a specified distance hurt performance.

   (a) Passage retrieval base

   This query expansion is rather experimental due to the time limitation.

   i. Extract passages from the top 15-20 documents with the following conditions

      A. Choose one passage of less than 2 KB

      B. The passage contains most of the terms in the query

   ii. Add 20-30 terms in the passage with weighting

      A. Order from most frequent to least frequent in the passages

      B. df of the terms in all document sets is more than 19 and less than 20000

      C. Use the Rocchio formula with weighting $\alpha = 6$,$\beta = 1$

   (b) Clustering base

   Our method of query expansion by document clustering consists of the three steps described below. All steps are implemented by EDA.

   First, we generated clusters of documents from the results of the pilot search. We compared several document-document relevance calculation methods and clustering methods implemented in KA and KAcluster, respectively. Each query took an average of three minutes.

   Second, we used the generated clusters to select good documents and bad documents. The rank of the pilot search was the basis of the documents evaluation for this selection. We called the rank of the pilot search RPS, hereafter. Document Selector provides several evaluation and manipulation techninuqes for a set of clusters and for clusters (sets of documents).

   Finally, we generated two sets of weighted words from good and bad documents based on document-word relevance. We compared several document-term relevance calculation methods implemented in KA. A set of weighted words from the bad documents was used to suppress inadequate words in a set of weighted words from the good documents. This suppression is very important for improving the precision of the final search. These sets of weighted words and the query for the pilot search (QPS) were combined into a query for the final search. Several parameters were combined. Each query took an average of two minutes.

   Our query expansion by document clustering offers many possible methods and many parameters. We combined several EDA functions by perl scripts, makefiles, and KAbatch scripts, and succeeded in implementing a flexible environment. Using qrels of TREC6 in order to build a query expansion by document clustering for TREC7, we selected a set of methods and determined a set of parameter values.

4. Final search

   We used OKAPI[16] for tf normalization. because we were not able to obtain good results with reference measure Lt.Lnu [11]. Besides our expermence in Lt.Lnu normaliztation, we were not able to obtain goot results with word frequency base text size normalization. We did not have enough time to determine the cause, but we suspect our handling of stemming or stop words.

4 Details

4.1 Measure

Our basic method was utilization of $tf * idf$, but with consideration of the following points:

- Words with small idf have a tendency to have high term frequency in a document, so the simple idf formula $log_2 N/n$ favors high df (small idf) words.

  A mecanically generated phrase with a large idf sometimes produces a bad result because its document frequency(df) is lower than 10, even as low as 1 or 2. Thus we established a minimal df.

- Measures favoring word co-ocurrence in a query get better results with short queries. To reflect this factor, we tested two solutions. One solution was to narrow the rage of tf in $tf*idf$; the other was to give bonus points when two words co-occur in text.

  In contrast, after automatic query expansion we need not consider the co-occurence of words in a query, because there are many words in the query.

- The text size of a correct text set for queries in TREC is mostly between one and three times of average text size. Measures that consider average text size, such as [16], [11], get better results.

1. idf

   $idf = log_2(\frac{N-\alpha*n}{n})$
   $idf = log_2(\frac{N-\alpha*N/k}{N/k}) if(n < l))$
   N = number of texts in the test collection
   n = document frequency for term
   $\alpha = 8 - 9$
   $l = 50$
   $k = 9000 - 10000$

2. tf of pilot search

   $tf1 = \frac{log_2(1+term\_freq)}{\sqrt[4]{\alpha*average\_doc\_size\_in\_byte+(1-\alpha)*doc\_size\_in\_byte}}$
   $tf = func(tf1, term\_freq, doc\_size\_in\_byte)$

   $func$ controls the range of $tf1$, and the max term num for each text size, and outputs tf value greater than 0.1, and less than 0.5

3. tf of final search (tf of OKAPI)

   $tf = \frac{(k_1+1)*term\_freq}{(k_1((1-b)+\frac{b*doc\_length\_in\_byte}{average\_doc\_length\_in\_byte})}$
   $k_1 = 1.5, b = 0.75$

4.2 Co-ocurrence Boosting

As was pointed out by [17],the measures favoring term co-ocurrence get better results for short queries. With a pilot search of a rather narrow tf range, we adopted a scoring method that favors term co-ocurrence.

Co-ocurrence boosting is implemented by simply multipling boost ratio to the similarity of each term.

$$S_i = \sum_t B * W_{t,i}$$

S_i is the degree of similarity between a document and topics.
i is the document number.
t is a term that document$_i$ includes.
$W_{t,i}$ is the part of similarity of term$_t$ in document$_i$.
B is the boost-ratio by term co-occurrence.

As expected, parameter B depends on the query. For example, in the TREC6 collection, one query gets the best result with $B = 3.0$, while another gets the best result with $B = 1.08$.

We did not have enough time to determine how to assign the best boost ratio for each query, so we set the parameter to 1.08 for short queries, and to 1.40 for very short queries through all the pilot search. We set the parameter to 1.00 for the final search.

4.3 MUST Operator Attachment

To raise the precision of the pilot search, we tested a new operator called the "MUST operator".

The MUST operator requires that the word specified by this operator must exist in the search results. In some cases, the precision of the pilot search increases when the MUST operator is used. This operator was tested to obtain higher precision in pilot search; it was not used in final search.

The use of the MUST operator consists of two steps. The first step is to select important terms. The second step is to retrieve documents that include the terms specified by the MUST operator.

We used the following simple rules to select important terms for specification by the MUST operator:

- Select a term that begins with capital character.
- Select two terms after the words "of", "in",and "the".
- DO NOT select adjectives (such as "*out", *and*" * *ble*").

Action by the MUST Operator is removed for words that fall under any of the following conditions:

- The df of the word is less than 25.
- The df of the word is more than 80000.
- The number of text hits becomes less than 100.

This rule, and the execution strategy of queries using the MUST operator is still tentative.

4.4 Query Expansion by Document Clustering

Our method of query expansion by document clustering consists of three steps: clustering the results of a pilot search, selecting documents, and generating a query for the final search. EDA supplies many functions for implementing these steps. In this section, we explain the real steps for TREC7.

4.4.1 Clustering pilot search results

We used KA and KAcluster with RP-1, RT-M, LtLnu, Inner Product, and Ward's method, then generated clusters from the top 1,000 documents resulting from the pilot search.

4.4.2 Selecting documents

Two sets of clusters (cluster sets) were generated from the generated clusters: one the good cluster set (GCS) and the other the bad cluster set (BCS). Because a cluster is a set of documents, a cluster set is a set of sets of documents.

The following steps were used to generate GCS and BCS. They were implemented by Document Selector.

1. Select clusters of an adequate size. We discarded every cluster that had fewer than n_1 elements or more than n_2 elements.
2. Generate GCS.
   (a) Measure clusters. For each cluster, we selected the best RPS as the representative document (REP) of the cluster, and calculated the average RPS of its elements (AVE).
   (b) Generate cluster sets. We grouped the clusters into cluster sets, each of which consisted of clusters with identical REPs.
   (c) Select each cluster from a cluster set. For each cluster set, we selected the cluster that had the best AVE in the cluster set. These clusters were combined to form a new cluster set whose elements have different REPs.
   (d) Select good clusters. From the cluster set, we selected n_3 clusters whose REPSs had better RPSs, and discarded all the other.
   (e) Select good documents. For each of the clusters in the good cluster set, we selected n_4 documents that had better RPS, and discarded all the others from the cluster. The result is GCS.
3. Generate BCS.

   We generated BCS in the same way as in Step 2, but 'good,' 'better,' 'best,' n_3, and n_4 in Step 2 were replaced with 'bad,' 'worse,' 'worst,' n_5, and n_6, respectively.

4.4.3 Generating a query

The query for the final search is generated from GCS, BCS, and QPS. A query is a set of words called a "word set," each of which has been assigned a weight.

The query for the final search consisted of the following steps:

1. Generate a good word set (GWS) from GCS.

   (a) Calculate weights. For each cluster in GCS, we used KA using RP-3, RT-M and LtLnu to generate a word set. The results were word sets, each of which corresponded to a cluster in GCS.

   (b) Select words. For each of the generated word sets, we selected n_7 words that had the greatest weight, and discarded all other words from the word set.

   (c) Average weights. For each word selected, we summed its weights in all word sets and divided the sum by the number of occurrences. The result is GWS.

2. Generate a bad word set (BWS) from BCS.

   We generated BWS from BCS as using a similar method to that in Step 1, except that n_7 is replaced by n_8 and the method of averaging weights in 1c is changed to summing the weight of each word in all word sets and dividing the sum by the square of its occurrence.

3. Combine GWS and BWS.

   For each word that is only in GWS, we multiplied its weight in GWS by n_9. For each word that is in both GWS and BWS, we multiplied its weight in GWS by n_9, multiplied its weight in BWS by n_{10}, and divided the former weight by the latter weight. The result is a word set that we call as the interim query (IQ). BWS contributes to suppress inadequate words in GWS.

4. Combine IQ and QPS.

   (a) Classify words. We divided the words in IQ and QPS into three groups: words only in IQ (OnlyIQ), words only in QPS (OnlyQPS), and words in both IQ and QPS (Both).

   (b) Recalculate weights for Both. For each word in Both, we multiplied its weight in IQ by n_{11} and added its weight in QPS to the multiplier.

   (c) Adjust weights. For each word in OnlyIQ, we multiplied its weight in IQ by n_{12}. For each word in OnlyQPS, we multiplied its weight in QPS by n_{13}. For each word in Both, we multiplied its weight, calculated in 4b, by n_{14}. The result was a new word set that consisted of all words in IQ and QPS with the multiplied weights.

5. Select words.

   we selected n_{15} words that had the greatest weight, and discarded all other words from the word set. The result is the query for the final search.

6. Comments

   Fifteen parameters $(n1, n2, \ldots, n15)$ appear in our query expansion by document clustering for TREC7. We decided on a set of values for the parameters by qrels of TREC6.

   At the step of adjusting weights in Step 4c, the multiplier of Both (n_{14}) should be set to a value that is greater than the others (n_{12} and n_{n13}). This adjustment of weights favor good words in QPS over bad words, and improves precision in the higher ranks in the final search.

   It is important to select clusters from cluster sets with appropriate variety in the step described in Section 4.4.2 (Selecting Documents) if the step described in Section 4.4.3 is to be processed successfully. If the clusters were selected only by AVE, the variety of the clusters is restricted. Step 2c is neccesary to solve the problem.

   Adequate values of all parameters depend on the deference of final search engines and documents sets. Our system implemented by EDA and some scripts is a flexible and powerful tool for determing a set of values for many search engines and document sets.

5 Ad hoc Results

Table 1: Eleven Point Average (Official Run)

Name	Flab7ad	Flab7at	Flab7atE
Category	Short	VShort	VShort
Mode	+CLS_QE +PH	+CLS_QE +PH	+MUST +QE +PH
at 0.00	0.7166	0.6771	0.6338
at 0.10	0.4962	0.4698	0.4530
at 0.20	0.3917	0.3635	0.3358
at 0.30	0.3298	0.2857	0.2644
at 0.40	0.2668	0.2206	0.2199
at 0.50	0.2055	0.1850	0.1796
at 0.60	0.1480	0.1370	0.1387
at 0.70	0.1006	0.0963	0.1052
at 0.80	0.0555	0.0579	0.0749
at 0.90	0.0287	0.0461	0.0402
at 1.00	0.0106	0.0154	0.0161
Average	0.2296	0.2126	0.2020

+CLS_QE = clustring-based query expansion; +MUST = Must operator;

+QE = passage retrieval base query expansion; +PH = phrase in query

Table 2: Eleven Point Average for Description (Base Point)

Category	Short	Short	Short	Short
Mode	-PH	+PH	-PH +QE	+PH +QE
at 0.00	0.7620	0.7734	0.6656	0.7103
at 0.10	0.4642	0.4516	0.4468	0.4811
at 0.20	0.3133	0.3340	0.3632	0.3801
at 0.30	0.2473	0.2753	0.2788	0.2974
at 0.40	0.1955	0.2138	0.2188	0.2372
at 0.50	0.1478	0.1504	0.1726	0.1935
at 0.60	0.1017	0.1091	0.1303	0.1475
at 0.70	0.0709	0.0810	0.0887	0.1151
at 0.80	0.0304	0.0505	0.0511	0.0652
at 0.90	0.0065	0.0169	0.0256	0.0330
at 1.00	0.0000	0.0046	0.0026	0.0132
Average	0.1863	0.1952	0.2042	0.2229

Table 3: Eleven Point Average for Title (Base Point)

Category	VShort	VShort	VShort	VShort
Mode	-PH	+PH	-PH +QE	+PH +QE
at 0.00	0.6806	0.6928	0.5806	0.6751
at 0.10	0.4162	0.4075	0.4442	0.4485
at 0.20	0.3056	0.2918	0.3683	0.3253
at 0.30	0.2315	0.2278	0.2902	0.2524
at 0.40	0.1740	0.1635	0.2375	0.2159
at 0.50	0.1322	0.1237	0.1696	0.1718
at 0.60	0.0892	0.0947	0.1259	0.1351
at 0.70	0.0587	0.0690	0.0921	0.1017
at 0.80	0.0380	0.0530	0.0532	0.0674
at 0.90	0.0131	0.0260	0.0296	0.0369
at 1.00	0.0011	0.0075	0.0040	0.0131
Average	0.1707	0.1706	0.2014	0.1993

Query expansion technique also produced a better result in our experiment. Clustering-based query expansion outperformed passage-based query expansion by 0.007 point.

Queries with phrases gets better results than queries without phrases, and queries with phrases also gets better results for query expansion. The phrase as a two-word pair within a specified distance without order sometimes results in documents including a bad phrase but with a high idf, and gets worse results than queries without phrase. We think that we must modify the phrase treatement in such cases.

Acknowledgement

The authors would like to acknowledge useful comments and suggestions of Mr. Isamu Watanabe who is the author of KA and Mr. Masayuki Sonobe of FUJITSU LABORATORIES LTD.

References

[1] Kunio Matsui, Isao Namba, and Nobuyuki Igata. A fulltext search system for large text (in japanese). *D-4-6*, 1997.

[2] Kunio Matsui, Isao Namba, and Nobuyuki Igata. High-speed text search engine (in japanese). *IPSJ 97-DD-7-3 pp15-21*, 1997.

[3] Niwa and Hirotora et al. Large text search system at altavista (in japanese). *IPSJ Advanded DataBase Symposium pp19-25*, 1996.

[4] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. Managing gigabyte - compressing and indexing documents and images. *Van Nostard Reinhold New York*, 1994.

[5] P. Elias. Universal codeword sets and representaions of the integers. *IEEE Transaction of Information Theory IT21:194-203*, 1975.

[6] William Pugh. Skip list a probablistic alternative to balanced trees. *CACM June 1990 Vol33 Num6 668-676*, 1990.

[7] Junichi Hagiwara, Tsunehisa Doi, Yoshinori Yaginuma, Kazuho Maeda, and Tatsuya Shindo. Commercial applications on the ap3000 parallel computer. *IEEE Massively Parallel Programming Models '97*, 1997.

[8] Tsunehisa Doi and Ikuo Miyoshi et al. Parallel text search engine on the ap3000. *Parallel Computing Workshop '97, P1-K*, 1997.

[9] Manabu Sassano and Isao Namba. A fast morphological analysis tool with user tuning (in japanese). *IPSJ 52th 5B-4 pp75-76*, 1996.

[10] Kozo Sugiyama, Kazuo Misue, Isamu Watanabe, Kiyoshi Nitta, and Yuji Takada. Emergent media environment for idea creation support. *Knowledge-Based Systems, 10, pp.51-58*, 1997.

[11] C. Buckley, A. Singhal, M. Mitra, and (G. Salton). New retrieval approaches using smart : Trec4. *TREC4 Proceedings*, 1994.

[12] Gerard Salton. Automatic text processing – the transformation, analysys, and retrieval of information by computer. *Addison Wesley*, 1989.

[13] Edie Rasmussen. Chapter 16, clustering algorithms. *Information Retrieval Data Structure and Algorithms ed. William B. Frakes, Ricardo Baeza-Yates Prentice Hall*, 1992.

[14] Porter M.F. An algorithm for suffix stripping. *Program, 13(3):130-137*, 1980.

[15] Chiristopher Fox. Chapter 7, lexical analysis and stoplists. *Information Retrieval Data Structure and Algorithms ed. William B. Frakes, Ricardo Baeza-Yates Prentice Hall*, 1992.

[16] S. E. Robertson, S. Walker, M. M. Beaulieu, and M. Gatford. Okapi at trec-4. *TREC4 Proceedings*, 1995.

[17] Ross Wilkinson, Justin Zobel, and Ron Sacks-Davis. Similarity measures for short queries. *TREC4 Proceeding*, 1994.