Fujitsu Laboratories TREC8 Report –Ad hoc, Small Web, and Large Web Track –

Isao Namba Nobuyuki Igata

Computer System Laboratory Fujitsu Laboratories Ltd.
{namba,igata}@flab.fujitsu.co.jp

Abstract

This year a Fujitsu Laboratory team participated in three tracks:that is ad hoc, small web track, and large web track. As basic techiniques, we compared four popular stemmers, and we made simple removing stop pattern techniques for TREC queries. For the ad hoc task, and small web track, we used the same techiniques. We experimented with area weighting, co-occurence boosting, bi-gram utlization, and reranking by bi-gram extraction from pilot search.

The effect of blind application with those techiniques is rather limited, or even uncertain in the TREC8 experiment. What we can say from TREC8 result is that blind application of co-occurence boosting and area weighting may be effective for the small web track. They requerie query dependent application.

In the large web track, our main interest is efficiency, that is how much resources are required to process 100GB of web text and 10000 real web queries in practical time. Using a statistical based language type checker, we can eliminate 23% of non-English text. This leads to speeding up a indexing and reducing the index size. The search speed for an inverted file is CPU intensive if the target machine has main memory in excess of 10-25% of the index size. So with simple, but effective index compression methods, the throughput of query processing is about 0.54-1.1 query/second even by a single 300MHz Ultra-sparc processor.

1 System Description

1.0.1 Teraß

Teraß[1] is a fulltext search library, designed to provide an adequate number of efficient functions for commercial service, and to provide parameter combination testing and easy extension for experiments in IR. For TREC8 we added functions for reranking pilot search results, and fixed bugs found during the TREC8 experiment.

1.0.2 trec_exec

trec_exec is designed for automatic processing of TREC. It contains a procedure controller, evaluation module , logging module, and all non-searching units such as query generation, query expansion and so on.

The motivation of making this prorgram is that we could not fully tune the system in the TREC7 experiment. In the TREC7 experiment it took about 5 minutes to evaluate the result, and we had difficulty in analyzing what parameter was actually effective as logging of parameters was imperfect.

trec_exec can execute all the TREC processing for one run in about one minute, and it can be used for system tuning by hill-climing.

2 Common Processing

The following process is common among all tracks.

Large web track does not use bi-gram, reranking by N-gram, and QE.

2.1 Indexing/Query Processing

2.1.1 indexing vocabulary

The indexing vocabulary consists of character strings made up of letters, numbers, and symbols, and no stop words were used in indexing. For TREC8, we modified the grammar of the token recognizer to accept acronyms with symbols such as U.S., and AT&T as one token.

2.1.2 Stemmer

We compared the popular stemmers. We selected four algorithms SMART[2], Porter[1] , Lovins[4], and Pickens[5] which are popular or can be used free experimentally. Their token recognizer was modified to compare in the same condition.

The table 1 is the result of a pilot search on the same parameter settings. The parameter tuning is based on the SMART stemmer and TREC7. Best average precision wass **bold face**, and worst was *italic*. T is the title field, D is the description field , and N is the narrative field. Blank field was not tested.

RUN	SMART	Porter	Lovins	Pickens
TREC4				
D	0.218,3351	*0.216*,3447	**0.221**,3364	
TREC5				
D	**0.163**,2125	0.158,2010	*0.149*,1935	
T	**0.151**,2050	0.146,1899	*0.144*,1684	
TD	**0.159**,1973	0.156,1989	*0.150*,1823	
TDN	**0.208**,2314	0.200,2360	*0.190*,2233	
TREC6				
T	0.211,2200	**0.218**,2215	*0.182*,2120	0.217,2182
D	**0.173**,1763		*0.158*,1631	0.170,1721
TD	0.237,2323	0.243,2320	*0.223*,2201	**0.244**,2324
TDN	**0.254**,2711	0.243,2581	*0.240*,2624	
TREC7				
T	**0.194**,2199	0.187,2227	*0.156*,2129	0.189,2182
D	**0.210**,2538	0.200,2392	*0.182*,2329	0.203,2391
TD	**0.237**,2323	0.215,2551	*0.194*,2479	0.219,2555
TDN	**0.257**,2762	0.238,2661	*0.230*,2638	0.256,2874

Table 1: Comparison of stemmers (average-precision,rel-ret)

[1]We found the official homge page of Porter Stemming Algorithm [3], and found that the author said almost all the implementation was different from the original one, and he made improvement of rules. Since we found the page in a few days ago, we did not try the official one.

Since the system is tuned based on the SMART Stemmer, and nothing is known about the relation between stemmer and system parameter tuning, it is difficult to deduce a concrete result.

What we feel is that SMART or Pickens are a better choice, and our choice is SMART.

2.1.3 Information in inverted file

Text number, term frequency, and term position are stored for the ad hoc task, and small web track for run time phrase processing and reranking by bi-gram extraction.

Only text number and term frequency are stored for the large web track to save disk space.

2.1.4 Stop word list for query processing

As in the TREC7[1], we used a stop word list of about 400 words of Fox[6], and words with a high df (more than 1/7 of the number of all documents) were also treated as stop words.

2.1.5 Stop pattern removal

The expression of TREC queries are artificial, so frequently appearing patterns such as "relevant document" are stop patterns. We generalized this observation, and removed the words which meet one of the following condition.

1. Word in stopword list is a stopword.
2. Word which is not a proper noun[2], and whose df in TREC1-7 queries is more than 400*0.1 is a stop word.
3. Word bi-gram whose df in TREC1-7 queries is more than 400*0.02 is a stop pattern.
4. Word tri-gram whose df in TREC1-7 queries is more than 400*0.01 is a stop pattern.
5. All the words in a sentence that contains "not relevant" are stop words.
6. 4 words following "other than" are stop words.
7. 4 words following "apart from" are stop words.

For the TREC-8, this parameter setting seems to remove too many patterns. The result is that official result is worse than with simple removal of stopwords about 0.5-1 point.

[2]U.S appears 94 times in TREC1-7 queries.

2.2 Weighting Scheme

The term weight is $qtf * tf * idf$, and the score for one document is the sum of the term weights with co-occurence boosting.

1. qtf

   qtf is the combination of the following parameters

   $qtf = \sum_f fw * tf * ttw$

   where

   f is the topic field (title, description or narrative).
   fw is weight of the topic field. We set the value for the title field to 3.0, the value for the description field 1.5, the value for the narrative is 0.9. Some teams [7], [8],[9] used weighting depending on field type, and we take the same approach.

   tf is the bare frequency in each field.

   ttw is the term type weight. It is set to 3 for terms, and set to 1 for phrase(word bi-gram).

2. tf

   We simply used the tf part of OKAPI[7].

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

3. idf

   We used a modified idf of OKAPI. We introduced a cut off point for low df words, and decreased the idf value for high df words.

   $idf = log_2 \frac{N-(n*\alpha)}{n}$
   N is the number of documents
   n is df if (df $>$ 1/10000 * N) else
   n = 1/10000 * N
   α is set to 3

2.3 Co-occurence Boosting

As in TREC7, we use co-occurence boosting techinique which favours co-occurence of query terms in a document. Co-ocurrence boosting is implemented by simply multipling the 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.

In the experiment of last year, we could not get improvment except in very short queries. One of the reasons was that applying this techinique to any symbols, that is word with high idf, word with low idf, and phrase(word bi-gram) caused the theme to drift. So we limited the application of this techinique only to words, and words within a limited df range.

Apparently, the best parameter B depends on the query, and we still have not found an automatic parameter control method. So we set the B to 1.10 for the title word, to 1.05 for the description word, and to 1.03 for the narrative word, and to 1.0 for the word added by query expansion.

2.4 phrase(bi-gram)

Instead of traditional IR phrase (two adjacent non-stopword pair with order or without order), we permitted limited distance in phrase. The motivation for introducing fixed distance is as follows. The first motivation is that non-stopword may exist between two adjacent words in a query. For example, in TREC7 query 351 "Falkland petroleum exploration", the word Island may be inserted between "Falkland" and "petroleum", but the two words may be located near each other in the sentence within a limited distance. The second motivation is that there are many stopword lists in the world and it is diffcult to select one ,and we don't like to remake the index every time we choose a different stopword set for an experiment, and stopwords are rather area dependent. The third motivation is that it is easy and fast enough to experiment by computing the frequency of the bi-gram using a word offset in an inverted file at run time.

The experimental environment is that the bi-gram uses bare idf, and its weight is 1/3 that of a normal word. The bi-gram is constructed from two adjacent non-stopword pair in a query which is not separted by stop pattern in stop pattern removal. Their source area is only the title and description fields. Table 2 shows the average precision after QE.

	bi-gram (dist=4)	phrase(no-order)
trec4 d	0.285	0.278
trec5 d	0.192	0.186
trec6 t	0.260	0.260
trec6 d	0.185	0.187
trec6 td	0.261	0.253
trec7 t	0.244	0.235
trec7 d	0.273	0.273
trec7 td	0.280	0.281
trec8 t	0.288	0.283
trec8 d	0.256	0.257
trec8 td	0.293	0.290

Table 2: Short distance bi-gram vs phrase

2.5 Reranking by best bi-gram

The bi-gram in query is limited to non-stopword pairs. This is because any bi-gram combination of terms in query easily drifts away from its original theme with its strong idf. [3] But in some cases, a non-adjacent pair in original query is a key expression. For example in TREC7 query 351 "Falkland petroleum exploration", the bi-gram expression of "Falkland" and "exploration" in rather narrow window (less than 50 words) is a good expression. All the 14 documents which contain this pattern are relevant. What we thought is frequently appearing bi-gram pattern of query terms in top ranked documents in the pilot search may contain such expressions, and can be used to make the document level average increase. Using TSV of Okapi, we selected bi-grams in rather middle size windows, and rerank the result of pilot search.

1. Rank top 1000 documents
2. Calculate the TSV score of every bi-gram of the terms in a query in fixed window size(set 40). The top 20 documents are supposed to be relevant, and the 500-1000 ranked document are supposed to be non-relevant.
3. Sort the bi-gram by TSV score
4. Select the best N (set 5) bigram whose mutual information is over the threshold(set -5). The threshold setting of mutual informaition means we accepted all the best bi-gram, and we did not consider the TSV score of the words in bi-gram.
5. Rerank the pilot search result after adding bi-gram score (tf*idf)

The effect of blind application of this techniques with this parameter setting and scoring seems to be dubious. Unexpectedly its effect on TREC4 is 10% and TREC5 5% and TREC7 3%, and we applied this techiniques to some of the official TREC8 runs.

The top ranked bi-gram expressions were the combination of main theme word(title word), and the other words in most cases. For example the selected bi-gram expression in TREC8 query 406 "Parkinson's disease" were "Parkinson disease", "Parkinson symptoms", "Parkinson treat", and "Parkinson patient".

Table 3 shows the change of document level average, and average precision after QE in this case.

Cond	@10	@20 (No QE)	av-prec(QE)
With	0.50	0.350	0.452
Without	0.40	0.350	0.371

Table 3: Document level average and average precision after QE for topic 406

2.6 Query Expansion

Query Expansion was used for the ad hoc task, and small web track. The Boughanem formula[7] was used to select terms.

$$TSV = (r/R - \alpha s/S).w^{(1)} \quad (1)$$

$w^{(1)}$ is modified and more general version of Robertson/Sparck Jones weight.

The α was set 0.001, and k4 was -0.3, k5 was 1, and k6 was 64. The top 20 documents in the pilot search were supposed to be relevant, and the documents ranked from 500 to 1000 were supposed to be non-relevant. The top ranked 40 words which are not included in original query, which are not included in the stopword list of SMART, whose tsv score are more than 0.003, whose df are more than 20, and whose df are less than 33000 were added to the original query.

No collection enrichment technique was used.

[3] We did not try Robertson's phrase weighting[7] at 1999/10/27

3 Ad hoc task

We tried many techiniques in the TREC8 ad hoc experiment, and most of them did not survive as their effects were uncertain or too merginal, and were just logged by trec_exec. As the following analysis shows, except QE, most of the techniques which we applied to the official runs were severely query dependent or target document dependent.

3.1 Ad hoc official runs

The results are shown in Table 4. TREC8 query was easier than that of TREC7, and there is little difference between the title only run and title and desciption run.

Run-id	Flab8as	Flab8atd2	Flab8atdn	Flab8ax	Flab8at
field	TD	TD	TDN	TDN	T
Av Prec	0.290	0.293	0.324	0.316	0.287
R-Prec	0.324	0.320	0.353	0.350	0.315
P@20	0.420	0.420	0.470	0.451	0.426
Retrieved	50000	50000	50000	50000	50000
Rel-ret	3090	2990	3261	3207	3084
Relevant	4728	4728	4728	4728	4728
best	0	2	1	1	3
>= med	42	39	39	41	40

Table 4: Eleven Point Average (Official Run)

The conditions of each run are given in table 5.

Name	Flab8as	Flab8atd2	Flab8atdn	Flab8ax	Flab8at
bi-gram	+	+	+	+	+
Co-boost	+	+	+	+	+
Rerank	-	+	-	-/+	+
QE	+	+	+	+	+
Data fusion	-	-	-	+	-

Table 5: Parameter condition of official Runs

Flab8ax is data fusion of Flab8atdn and very long query version of the Flabatd2 condition. For the TREC7 experiment, merging two different system results in 1-1.5 points up in some cases (eg. OKAPI+INQUERY). But as this result shows, it is not stable.

3.2 Ad hoc analysis

The effect of the techiniques we employed, field weighting, co-occurence boosting, bi-gram in query, and reranking by bi-gram are shown in the following tables.

bi-gram	field	boost	rerank	No QE	QE
-	-	-	-	0.243	0.271
+	-	-	-	0.247	0.288
+	+	-	-	0.251	0.290
+	+	+	-	0.252	0.291
+	+	+	+	0.241	0.293 (*official)

Table 6: The effects of each techniques (title and description)

bi-gram	field	boost	rerank	No QE	QE
-	-	-	-	0.215	0.259
+	-	-	-	0.236	0.288
+	+	-	-	0.236	0.289
+	+	+	-	0.238	0.290
+	+	+	+	0.235	0.288 (*official)

Table 7: The effects of each techniques (title)

4 Small Web Track

The processing of Small Web Track is the same as that of the ad hoc task.

No link run is submitted as we did not have enough time, and it seems difficult to use link information for 2GB text.

What we felt during system training by TREC7 queries was the searching for small web data easily drifted away from the main theme.

The result seems to support our feelings, that is field weighting and co-occrence boosting is the most effective of our techniques, and QE is less effective than in the case of the ad hoc task.

4.1 Small Web Track official Runs

Two runs are submitted, Flab8wtdnN and Flab8wtdN. Their procedure was the same as that of Flab8atd2. Flab8wtdnN used full field, and

Flab8wtdN used title and description field. The results are shown in Table 8.

Name	Flab8wtdnN	Flab8wtdN
field	TDN	TD
link	NO	NO
Average Prec	0.340	0.332
R-Prec	0.353	0.355
P@20	0.401	0.398
Retrieved	50000	50000
Rel-ret	2279	2279
Relevant	1988	1954
best/ >= med	5/42	1/42

Table 8: Official small web track result

4.2 Small Web Track analysis

The following two tables show the combination of each techinique.

bi-gram	field	boost	rerank	No QE	QE
-	-	-	-	0.289	0.325
+	-	-	-	0.296	0.323
+	+	-	-	0.304	0.338
+	+	+	-	0.315	0.340
+	+	+	+	0.301	0.332 *official

Table 9: The effects of each technique (title and description)

bi-gram	field	boost	rerank	No QE	QE
-	-	-	-	0.294	0.321
+	-	-	-	0.306	0.332
+	+	-	-	0.323	0.347
+	+	+	-	0.331	0.358
+	+	+	+	0.322	*0.346

Table 10: The effects of each technique (title description,and narrative)

* official result is 0.340.

5 Large Web Track

Our goal for the large web track is to show that a single CPU is enough for processing 100GB of Web data, and even a slow CPU (Ultra-Sparc 300MHz) is enough. This is because inversion and searching is CPU intensive if hardware conditions are met, and such hardwares are not so expensive today.

The balance between speed, precision and hardware cost was our research goal.

5.1 Hardware environment

One Sun Ultra2 workstation was used for the lage web track. It has 1GB memory and has 2 Ultrasparc 300MHz CPUs. Most of the processing was done using 1 CPU, sometimes 2 CPU were used.

5.2 islang

As pointed out by some groups[7], large web track data contains considerable non-English data. This data increases the size of the index, and slows down the inversion, and is never retrieved in English query processing.

A statistical based language type checker, called islang, which rejects both non-English text, and non-language text was originally designed to select Japanese and English text for Web search engine[10].

The basic idea of islang is that frequently appearing spelling, prefix and suffix patterns are the key expression in language type checking, and we can use the tri-gram as the pattern in English.

The algorithm is as follows.

1. Training
   (a) L is the training corpus containing N patterns.
   (b) The pattern is the tri-gram of case folded alphabetical ascii character.
   To reflect the word construction of English, patterns in word and patterns adjacent to word boundaries are treated as two different patterns.
   (c) Simply count the frequency of patterns and calculate the information of each patterns in the corpus.
   $n_i (i = 1, 2...n)$ is the pattern in L
   f_i is the frequency for n_i respectly.
   $p_i = -log_2(\frac{f_i in L}{N})$
2. Checking
   (a) T is the target text containing M patterns.

(b) $t_i (i = 1, 2...n)$ is the pattern in T.

f_i is the frequency for t_i respectly.

(c) Calculate average information and score.

$av = (\sum_{i=1}^{i=n} f_i * p_i \ for \ t_i)/M$
If t_i is not found in n_i, use $\alpha * -log_2(1/N)$ instead of p_i
socre is 2^{av}

(d) If score is over the threshold, reject text as not belonging to the language.

α is set to 1.0, and threshold is set to 40000, and TREC7 documents are used as a training corpus.

The example of rejected documents from IA001-0000-B001- set are given in Table 11.

DOCID	score	text
5	44167	Die Dapsy Dinos Leider wurde
44	81499	Sydsvenskan - Nyhetsrsikt ...
52	51064630	323c 2f44 4f43 4e4f 3e0a ...

Table 11: Sample of rejected document by islang

23% of the text is rejected as non-English. The sum of word entry in inverted file is reduced to 10 million from 20million, and the index size is reduced to 4.0GB from 4.8GB without stopword condition.

5.3 Large web track result

Our main concern is the balance between performance and precision. Introducing Boolean AND condition speeds up query processing, but precision may go down. To compare simple ranking and AND condition with ranking, we submitted two runs. All runs did not use phrases, and query expansion. B+R means ranking document with AND condition of every non-stopword in a query. If the number of retrieved documents is less than 20, then ranking search is retried. This AND conditional interface is popular in actual Internet services. R means traditional accumulator method. Only fl8wlsb used index with stopwords. Table12 shows our official result.

Run-id	av-prec	P@10	P@20	Calc	speed(sec)
fl8wlnsb	0.4103	0.536	0.510	B+R	0.75
fl8wlnsr	0.4064	0.538	0.508	R	1.16
fl8wlsb	0.4116	0.540	0.507	B+R	0.54

Table 12: Large web official result

There is no remarkable difference in precision. B+R search and index with stopwords seems to be the best choice considering speed.

5.4 Performance of pre-processing

The preprocessing involves web detagging, running islang, and indexing. The official pre-processing data is as follows.

1. Detagging etc. took 3 or 4 weeks using 1 CPU.

   The process contains gzip, gunzip, cat, copy and rm for original data due to shortage of disk space, But most of the time was comsumed by a poor detagging script. After detagging text size is reduced to about 50GB.

   After submission of result, we wrote C version of detagging program. Its processing time was about 10 hours including the time for gunzip the data.

2. islang takes 5.23 hours using 2 CPU.

   23% of the text was rejected as non-English.

3. Indexing

   It took about 30 hours using 1 CPU.

   To compare the effect of the stopword, we made two indexes. 'With stopword' uses the stopword list of SMART[2], and rejects non-alphabetical symbols, digital string equal or greater than 1000000, and strings longer than 24 characters.

   The official inversion is very slow because we failed to keep working area(300MB) of memory. Solaris2.6 swapped out the working area while inversion was being done. So more than half of the time was comsumed by page-in and page-out. After submission of the official result, we found we could lock the work area in memory by mlock(), and mincore() system calls in Solaris2.6, but we did not retry by now(1999/10/27).

condition	time	status	work area
With stopword	30.38 hours	official	300MB
Without stopword	27.63 hours	official	300MB

Table 13: Inversion time

The Index size is given in table 14.

files	with stopword	without stopword
inverted file	3.01	4.03
dictionary	0.46	0.59
text size array	0.07	0.07
text number id	0.41	0.41
total	3.95GB	5.10GB

Table 14: Index size

5.5 Performance of query processing

In this section we show the basic figure of query processing.

5.5.1 Design of Teraß

We briefly list the features of Teraß concering performance.

1. I/O optimization
   (a) simply locating the bitmap entry of an inverted file for the same term in continuous area.
   (b) Clustering frequently accessed bitmap entries.(not used in TREC run)
2. Index entry is compressed by extended gamma coding [11] which aims at a balance between index size and decoding speed.
3. A Skip list [12] is used to minimize decoding cost.
4. Many code optimization techniques are used such as loop expansion, macro expansion, and other coding techniques.

5.5.2 Average Processing Speed

The regulation of a large web track says that query processing speed is the total processing time divided by the number of query. Other teams generally take the data parallel model to speed up processing, so we took the round-robin model to increase speed. The processing is very simple. We split a query file into 2 or 3 files, and run 2 or 3 search programs to the same index. 1, 2, and 3 processes are compared and the result is given in Table 15.

Run-id	1 process	2 process	3 process
fl8wlnsb	0.75	0.40	0.53
fl8wlnsr	1.16	0.75	0.87
fl8wlsb	0.54	0.39	0.63

Table 15: Query processing speed (elapsed seconds per query)

As the result shows that 2 process is the fastest in 2 CPU environment. This suggest searching is CPU intensive for some conditions, and we show the evidence in the following secitons.

5.5.3 CPU time vs real time

If CPU time is dominant in most of the searching processes, there is little difference between the distribution of CPU time and real time. The histogram of query processing speed is figure 1 . This speed only includes searching the inverted file.

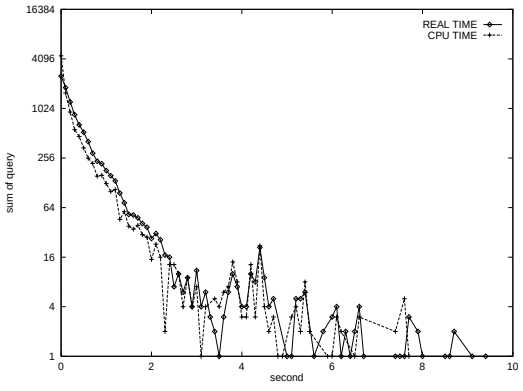

Figure 1: CPU time vs real time

For most of the queries CPU time is nearly equal to real time. But in the case of huge queries they differ.

Figure 2 shows the relation between the number of terms in query and average processing time.

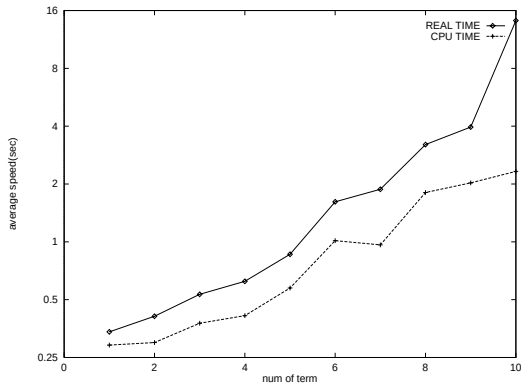

Figure 2: CPU time vs real time

The shortest CPU time is less than 1/1000 second, and the longest CPU time is 7.7 seconds. The shortest real time is less than 1/1000 second, and the longest real time is 28.1 seconds.

5.5.4 Index size vs available Memory

What actually affects the search performance is memory pages which the OS caches, and not search system caches if the index uses Unix File System[4]. But it is difficult to know what block of what files the OS caches. So we limit avaliable main memory size using a memory resident program using mlock() and mincore(), and check search program performance.

The result is omitted in this paper, but we get the same result as that of past experiment. In our past experiments on patent abstract (index size about 1GB), the query processing was CPU intensive if memory size is more than 10-25% of index size. In our past experiment with Japanese patent data (index size 10GB), and actual query (about 13000 queries), the total of the accessed inverted file entry is about 1GB.

5.5.5 More speed

We simply list the techiniques for increasing speed.

[4]This means not using raw disk

1. inversion
   (a) Stemmer
   The speed of the stemmer is given in Table 16.

Stemmer	speed(Gbyte/hour)
No stemmer	16.5
Porter	6.64
SMART	6.36
Lovins	7.10

Table 16: Stemmer speed comparison

   Almost all stemmer implementations are run time rule matching, so they are not fast from the view of implementaion.
   (b) Sorting in sort merge inversion
   Teraß uses a sorting merge algorithm for inversion, and quick sort algorithm is used to sort the entries of an inverted file. But the quick sort algorithm is not the fastest in this special case.

   Stemming and sorting occupies more than 70% of our inversion process.
2. Searching
   (a) I/O optimization
   Though the main memory size is large enough compared with index size, inverted file entry which is actually accessed in query processing is not clustered.
   (b) Changing Measures
   We used OKAPI for all TREC8 runs. The tf calculation of OKAPI requires division every time as its form is $tf/(tf+\alpha)$. It slows down the searching speed. To the contrary, the vector space model measures such as dtn.dnb [13] require division only once for each document. Table 17 shows the CPU time of OKAPI and dtn.dnb weighting.

Measure	CPU	real
OKAPI	0.353	0.496
ddb.ddn	0.270	0.410

Table 17: Measure speed (elapsed second per query)

6 Conclusion

Though the combination of above techniques seems to be working for automatic ad hoc, and small web track, each technique is serverly query/search result dependent. We need control method whether we apply those techniques or not. The result on the Large Web track is good. The balance of indexing speed, index size, searching speed, and precision is satisfactory, considering the hardware resources, which is not measured by price because current PC is faster than old workstation, and is cheaper.

References

[1] I Namba, N Igata, H Horai, K Nitta, and K Matsui. Fujitsu laboratories trec7 report. *The Seventh Text REtrieval Conference*, 1999.

[2] SMART ftp cite. ftp://ftp.cs.cornell.edu/pub/smart/. 1999.

[3] Martin Porter. Official homge page of porter stemming algorithm. *http://www.muscat.com/~martin/stem.html*, 1999.

[4] MG Home Page. http://www.mds.rmit.edu.au/mg/welcome.html. 1999.

[5] Jeremy Pickens. Stemming and cooccurrence on a larger corpus. *http://ciir.cs.umass.edu/*, 1999.

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

[7] S E Robertson, S Walker, and M Beaulieu. Okapi at trec-7. *The Seventh Text REtrieval Conference*, 1999.

[8] D R H Miller, T Leek, and R M Schwarts. Bbn at trec-7. *The Seventh Text REtrieval Conference*, 1999.

[9] James Allan, Jamie Callan, Mark Sanderson, Jinxi Xu, and Steven Wegmann. Inquery and trec-7. *The Seventh Text REtrieval Conference*, 1999.

[10] InfoNavigator. http://infonavi.infoweb.or.jp/. 1999.

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

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

[13] Amit Singhal, John Choi, Donald Hindle, David D. Lewis, and Fernado Pereia. At&t at trec7. *The Seventh Text REtrieval Conference*, 1999.