AT&T at TREC-6

Amit Singhal
AT&T Labs–Research
singhal@research.att.com

Abstract

TREC-6 is AT&T's first independent TREC participation. We are participating in the main tasks (adhoc, routing), the filtering track, the VLC track, and the SDR track[1] This year, in the main tasks, we experimented with multi-pass query expansion using Rocchio's formulation. We concentrated a reasonable amount of our effort on our VLC track system, which is based on locally distributed, disjoint, and smaller sub-collections of the large collection. Our filtering track runs are based on our routing runs, followed by similarity thresholding to make a binary decision of the relevance prediction for a document.

1 Introduction

TREC-6 is the first TREC in which AT&T is participating as an independent group. Much of our work is largely inspired by Smart's philosophy of fully automatic processing of large text collections. Our participation is based on an internally modified version of Cornell's SMART system. We submitted runs for the adhoc task, the routing task, the filtering track, the VLC track, and the SDR track (see footnote 1).

In the main tasks, the highlight of our preparation for TREC this year was our repeated failure to improve upon Cornell's TREC-5 results (we were a part of Cornell's TREC participation last year). In the routing task, we tried many new techniques, and variations of old techniques, but nothing provided a noticeable improvement in performance over last year's results. We finally settled for a two-pass query modification algorithm, with the second pass intended to fix the weakness of the first-pass query. This yields small improvements in our routing performance. In the adhoc task, we augment the "goodness" of a query-term by a new "importance factor" in addition to the usual query term weight, for selecting the top documents to be used in pseudo-feedback.

2 Routing Runs

Our routing runs use routing queries learned in a query zone using Rocchio's formulation. [7] All term weighting in our system is based on pivoted-unique document length normalization. [6] The first official run, **att97rc** (routing, conservative), uses the routing algorithm presented in Table 2.

Unfortunately, **our official run att97rc has a bug** that resulted in non-optimized word-pair weights. Fixing it improves the performance reasonably. Table 3 shows the results of our buggy official run **att97rc**, as well as the results when the bug is fixed. Since the fixed run was not in the pool of runs used to compute the best/median statistics, we notice that the fixed run is actually better than the best official result for three of the topics, and is above median for 46 out of 47 topics. These numbers suggest that the above routing algorithm is quite effective.

Table 4 shows the effectiveness of various components of the above routing algorithm. When no query zoning is used, *i.e.*, all non-relevant articles are used in Rocchio's formula, a different set of Rocchio parameters ($\alpha = 8$, $\beta = 64$, $\gamma = 256$) is known to be more effective [7], and we obtain an average precision of 0.3296. Once we switch to using query zones, we obtain a 8% improvement over not using query zones. This is in strong agreement with our earlier experiments on other TREC routing tasks. [7] Now we can either optimize

[1] This report does not describe our SDR track participation. Please see the adjoining report "AT&T at TREC-6: SDR Track" for details of our SDR system.

l tf factor:

$$1 + log(tf)$$

L tf factor:

$$\frac{1 + log(tf)}{1 + log(average\ tf\ in\ text)}$$

t idf factor:

$$log(\frac{N+1}{df})$$

u length normalization factor:

$$\frac{1}{0.8 + 0.2 \times \frac{number\ of\ unique\ words\ in\ text}{average\ number\ of\ unique\ words\ per\ document}}$$

where, *tf* is the term's frequency in text (query/document)
N is the total number of documents in the training collection
df is the number of documents that contain the term, and
the average number of words per document is 110.

ltu weighting: **l** factor $\times$ **t** factor $\times$ **u** factor
Lnu weighting: **L** factor $\times$ **u** factor
Ltu weighting: **L** factor $\times$ **t** factor $\times$ **u** factor

Table 1: Term Weighting Schemes

the query without adding word-pairs, or after adding word-pairs. If we optimize the query without adding word-pairs, we get an overall improvement of about 16% over our baseline. But if we do add word-pairs (as explained in step 3 of the algorithm in Table 2), prior to optimization, just by adding 100 pairs, we get an improvement of about 13% over the baseline. Optimization of pair-added queries yields even richer improvements than optimizing the non-pair-added queries, yielding an overall improvement of about 25% over our baseline.

The above routing algorithm is quite similar to the routing algorithm we used in TREC-5 [2], except for minor variations. We tried various new techniques to improve upon the above routing algorithm, but none of the techniques we tried yielded better results that the above algorithm.

Our first approach revolved around clustering the known relevant articles for a query. The main thought behind this approach was that relevance can have "multiple aspects". For example, for a query on *trade barriers in Japan*, one aspect of the relevant documents is trade barriers in the automobile industry (with keywords like: *chrysler, ford, mitsubishi, ...*), yet another aspect is trade barriers in the electronics industry (with keywords like: *toshiba, sony, ...*). If one can isolate such patterns in the relevant documents, it should be possible to learn one query per aspect and this query should be better than one global query for routing documents related to that aspect. Unfortunately we were unable to improve our routing performance using such an approach, mainly, we believe, due to the following reasons: a) not too many queries have clearly defined multiple aspects of relevance, b) once we cluster documents and select an aspect, the amount of training data (relevant and non-relevant documents) is much less for the aspect, resulting is a poorer feedback query; and c) a good single query already incorporates the multiple aspects of relevance in it, for example, the feedback query for the above example will have keywords from all aspects (*chrysler, ford, mitsubishi, toshiba, sony, ...*), thereby implicitly giving us the benefits we had hoped to obtain from clustering.

The second approach we tried was based on using a multi-pass query refinement technique. The basic idea behind this scheme is to compensate for the deficiency of a feedback-query, by enhancing it with another pass of feedback. For example, once we learn a first pass query using Rocchio's formulation (no optimization), we can use this feedback query to rank the *training* collection. This feedback query will rank some non-relevant documents at top ranks. These are the non-relevant documents that the first pass feedback-query is having difficulty "defeating". If we learn another query specifically aimed at defeating these non-relevant documents

1. Using *ltu* weighted queries (see Table 1), and *Lnu* weighted *training documents*, form a training "query-zone" by retrieving the top 5,000 documents for the query (using the inner-product similarity).

2. Using the non-relevant documents in the query-zone, and *all* the relevant documents in the training corpus, form a feedback query using Rocchio's formulation using the following constraints/parameters:

   - Document terms are *Ltu* weighted. Original queries are *ltu* weighted.
   - Only the original query terms, and the "non-random" words and phrases, *i.e.*, the words that appear in at least 10% of the relevant articles, and phrases that occur in at least 5% of the relevant articles are considered for use in the feedback query.
   - Top 100 words and 20 phrases, as weighted by the Rocchio formula:

     $$8 \times \textit{original query vector} + 64 \times \textit{average relevant vector} - 64 \times \textit{average nonrelevant vector}$$

     are retained in the feedback query with weights predicted by the above formula. The average relevant vector is the average vector of all the relevant documents: $\frac{1}{|R|} \times \sum_{D_i \in Rel} \vec{D_i}$, where $|R|$ is the number of known relevant documents. The average non-relevant vector is defined correspondingly.

3. The query formed in the above step is a recall-oriented query. To enhance the precision of the query, we add query-word—query-word cooccurrence pairs to the above query. If two words occur *in the same document*, they form a potential cooccurrence pair.

   - Using the 100 query words from the previous step, we consider the 4,950 word-pairs.
   - All the "random" word-pairs, *i.e.*, the word-pairs that occur in fewer than 7% of the relevant documents, are removed.
   - Since we want to add a precision tool to the query, we re-sample the training non-relevant documents, and use the *top* $2 \times |R|$ non-relevant documents from step 1. Here $|R|$ is the number of training relevant documents.
   - Using *all* the relevant documents, and this restricted set of non-relevant documents (a tighter query-zone, so to speak), we add to the query (from step 2) the 100 word-pairs with highest weights as weighted by the following Rocchio formula:

     $$64 \times \textit{average relevant vector} - 64 \times \textit{average nonrelevant vector}$$

   Since word-pair weights in documents are needed in the above formula, to compute the *Ltu* weight for a pair, the lower of the constituent words' *tf* is considered as the pair's *tf*. Pair *idf* is computed on the fly by computing the true pair *df* by intersecting the individual words' inverted lists.

4. Term weights in this query of 100 words, 20 phrases and 100 word-pairs are further optimized using three-pass dynamic feedback optimization (DFO) with pass ratios 1.00, 0.50, and 0.25. [1]

5. The optimized feedback query is used to rank the new (test) documents. The test documents are *Lnu* weighted (see Table 1).

Table 2: Routing Algorithm

Run	Average Precision	> Best	Best	>= Median	< Median
Official (buggy)	0.3963	–	4	43	4
Fixed	0.4132	3	0	46	1

Table 3: Results for **att97rc**

	No QZ $\alpha.\beta.\gamma : 8.64.256$	QZ $\alpha.\beta.\gamma : 8.64.64$	QZ+DFO (No Pairs)	QZ+Pairs (No DFO)	QZ+Pairs+DFO
Avg. Prec	0.3296	0.3560	0.3819	0.3716	0.4132
Improvement (over No QZ)	–	+ 8.0%	+15.9%	+12.7%	+25.4%

Table 4: Effect of various components of **att97rc**

Run	Average Precision	> Best	Best	>= Median	< Median
Official (buggy)	0.4207	–	4	45	2
Fixed	0.4307	3	0	45	2

Table 5: Results for **att97re**

(using Rocchio's formulation with all the relevant documents and these top few non-relevant documents), then by combining the first pass and the second pass query, we should be able to get an overall improved query. We found that such two-pass approach improves routing effectiveness in experiments on the TREC-3, 4, and 5 routing tasks, over using a single pass non-optimized feedback. But the resulting two pass query is still somewhat poorer than the optimized one pass query. Optimizing the two pass query didn't buy us much. Overall it is a wash to use a multi-pass query or a single pass optimized query.

A minor variation of the above multi-pass approach did yield very small improvements over an optimized one pass query for the TREC-3, 4, and 5 tasks, and was submitted as our other official run **att97re** (routing, experimental). The idea in this run is to find the relevant documents that the first pass feedback-query is not ranking well in the training collection, *i.e.*, the bottom ranked training relevant documents (as ranked by the feedback-query), and the non-relevant documents that the first pass query is not defeating well, *i.e.*, the top ranked training non-relevant documents. This idea bears resemblance to the class of algorithms known as boosting in the machine learning community. [3] We select the bottom $|R|/2$ relevant documents, and the top $2 \times |R|$ non-relevant documents (where $|R|$ is the number of training relevant documents for a query).

We take the query formed using steps 1-3 of the algorithm in Table 2, rank the training collection using this query, and select the bottom $|R|/2$ relevant documents, and the top $2 \times |R|$ non-relevant documents. We independently form another query of 100 words, 20 phrases, and 100 word-pairs using these training documents (using steps 2-3 of the algorithm in Table 2). The final query is constructed using the following formula: pass-1 query + 0.25×pass-2 query. This final query is then optimized using a 3-pass DFO (as in step 4 in algorithm in Table 2). Unfortunately our official submission **att97re also has a bug**. The phrase and cooccurrence contributions were reduced (0.5 used in place of 1.0) due to a bug in the shell script used in **att97re**. Once the bug is fixed, the average precision for **att97re** improves some. This run is about 4% better than our conservative run **att97rc**. Table 5 shows the results of **att97re**. We believe that such multi-pass approaches for routing are promising, and deserve a more careful study.

Aside

In doing some post hoc analysis of where our current routing algorithms are failing, and why aren't we observing any marked improvements in the best routing effectiveness over the last few TRECs, we read several documents retrieved at top ranks by our routing queries. While reading through these documents, we did find many instances where a non-relevant article was ranked high because of the limitations of the statistical nature of our systems. But often enough, we found ourselves wondering why a document was judged non-relevant while another, very similar document was judged relevant. For the adhoc task, on reading the documents, it was much more obvious to us why documents were judged relevant or non-relevant. Voorhees and Harman report a three-way assessor agreement rate of approximately 72% for the adhoc task, [8] which is a very respectable agreement rate. We wonder if this figure would be lower for the routing task. It would be interesting to do such an assessor agreement study for the routing task, especially since the documents in the judgment pool are being retrieved by queries that have been learned using a large amount of training data, and are therefore much more precise (or effective) than the adhoc queries.

Word	df in 1,000	df	$\frac{df\ in\ 1,000}{df}$	Weight	Factor	Final Weight
hazard(ous)	386	7125	0.0542	5.34903	1.0000	5.34903
termin(als)	444	11903	0.0373	4.71901	0.6838	3.22673
comput(er)	454	22505	0.0202	3.93704	0.5528	2.17634
health	561	43015	0.0130	3.14174	0.4523	1.42094
daily	262	21034	0.0125	4.02003	0.3675	1.47754
individual(s)	474	44335	0.0107	3.10463	0.2929	0.90933
basi(s)	427	42023	0.0102	3.17038	0.2254	0.71461
work	617	148250	0.0042	1.62267	0.1633	0.26505

Table 6: Term Ordering for Topic 350

3 Adhoc Runs

Over the last few years, it has been shown that pseudo-feedback, *i.e.*, query modification without any relevance feedback from a user, *assuming* that the top few documents retrieved by the user query are relevant, yields noticeable improvements in retrieval effectiveness in the adhoc task. [4, 8] Typically we have been using the top twenty documents retrieved by the original query for pseudo-feedback. Motivated by Hearst's observations in [5], recently we have tried improving the quality of our relevance assumption by reranking the top fifty documents retrieved by the original query according to some "precision criteria" and using the top twenty documents from this reranked list in pseudo-feedback. [2] One particular criteria that we have used is the presence of several query terms in a small window of text in a document (see Table 7).

This year, we used a new method to rerank the top fifty documents to select the set of twenty documents used in pseudo-feedback. This technique is based on a new query term weight modification factor that we use to assess the importance of a query term in addition to the regular query term weight *ltu* (see Table 1). During experimentation, we observed that the goodness of a query term is related to the number of documents, in the top (say, 1,000) documents retrieved by the query, that contain the term. But since common words can appear in many documents, we need to normalize the above measure by the global df of the term. We used the following function to rank the original query terms:

$$\frac{number\ of\ documents\ in\ the\ top\ 1,000\ documents\ (retrieved\ by\ the\ query)\ that\ contain\ the\ term}{number\ of\ documents\ in\ the\ collection\ that\ contain\ the\ term\ (df)}$$

For example, for query 350, "Is it hazardous to the health of individuals to work with computer terminals on a daily basis?", the term ordering generated by this scheme is shown in Table 6. The query words are listed in decreasing order of their perceived importance in Table 6. This method does rank terms that we intuitively know are most important (*e.g.*, hazard, terminal) ahead of other terms that we think are less important (*e.g.*, basis, work). Even though a purely idf based ranking will place a relatively less useful word, like *basis*, ahead of a more useful word, like *health.*

After the terms in a query are ranked by the above formula, their weights are modified by multiplying with the following importance factor:

$$1.0 - \sqrt{\frac{rank - 1}{10}}$$

This factor lowers the weights of the terms ranked poorly in the above ranking, thereby emphasizing the top few terms noticeably. Table 6 also shows the original query term weight (column 5), the value of the above factor for a term (column 6), and the final query term weight for reranking the top fifty documents (column 7). We can see how less important terms, like *basis and work*, get a very low final weight. By using this weight modification factor, we ensure that a combination of the low ranked (hopefully less useful) terms will not defeat a presence of a high ranked term, which is often essential for relevance. Table 7 shows our full adhoc algorithm.

Table 8 show the performance of the various components of our adhoc algorithm over several TREC tasks. We use only the **description** field of the queries for the results reported in Table 8. The second column has the results for a straight vector run. The third column shows the results when the top 20 documents

1. Retrieve 1,000 documents using *ltu* weighted queries and *Lnu* weighted documents.
2. Rerank the query terms by $\frac{df\ in\ 1,000}{\mathrm{df}}$ and multiply their weight by $1.0 - \sqrt{\frac{rank-1}{10}}$.
3. Using the re-weighted query, rerank the top 50 documents. Documents are broken into 50 words overlapping windows starting at every 25th word, and a document's score is the best score of any window in that document.
4. Top 20 documents in this reranked list are *assumed* relevant. Since majority of the bottom ranked documents are usually non-relevant, documents ranked 501 to 1,000 are *assumed* to be non-relevant. Pseudo-feedback is performed using these assumptions, and the query is expanded by 25 words and 5 phrases. (Rocchio parameter values of $\alpha = 8$, $\beta = 8$, $\gamma = 8$ are used.)
5. The expanded query is used to rank the collection to get the final ranking for documents.

Table 7: Adhoc Algorithm

Task	No Feedback (Lnu.ltu)	Top 20 Relevant	501-1,000 Non-Relevant	Rerank based on locality
TREC-3	0.2385	0.3214	0.3340	0.3462
	–	+34.8%	+40.1%	+45.2%
TREC-4	0.2303	0.3000	0.3082	0.3167
	–	+30.2%	+33.8%	+37.5%
TREC-5	0.1505	0.1855	0.1909	0.2010
	–	+23.3%	+26.8%	+33.5%
TREC-6	0.1621	0.1723	0.1849	**0.1847**
	–	+ 6.3%	+14.1%	+14.0%

Table 8: Effect of various pseudo-feedback methods on adhoc performance, description-only queries. Our official run **att97ac** is shown in bold.

Task	No Feedback (Lnu.ltu)	Top 20 Relevant	501-1,000 Non-Relevant	Rerank based on locality
TREC-6 Title-Only Queries	0.2005 –	0.2017 + 0.6%	0.2079 + 3.7%	**0.2289** +14.2%
P@20	0.3070 –	0.3200 + 4.2%	0.3210 +4.6%	0.3530 + 15.0%

Table 9: Effect of various pseudo-feedback methods on title-only TREC-6 adhoc queries. Our official run **att97as** is shown in bold.

Query	Query Length	No Feedback (Lnu.ltu)	Top 20 Relevant	501-1,000 Non-Relevant	Rerank based on locality
Title Only (T)	3.02	0.2005	0.2017 (+ 0.6%)	0.2079 (+ 3.7%)	0.2289 (+14.2%)
Title+Desc (T+D)	11.78	0.2064	0.1931 (– 6.5%)	0.2071 (+ 0.3%)	0.2237 (+ 8.4%)
Improvement over T		+ 3.0%	– 4.3%	– 0.4%	– 2.3%
Full (T+D+N)	33.60	0.2179	0.2210 (+ 1.4%)	0.2282 (+ 4.7%)	0.2384 (+ 9.4%)
Improvement over T		+ 8.7%	+ 9.6%	+ 9.8%	+ 4.1%

Table 10: Performance of different lengths of TREC-6 adhoc topics.

from the straight vector run are assumed to be relevant and pseudo-feedback is performed. The fourth column also assumes documents ranked 501-1,000 as non-relevant (in addition to the third column). The fifth column is the reranking run (in addition to assuming 501-1,000 non-relevant). It is evident that pseudo-feedback improves performance across tasks. However, we should note that the improvements obtained for this year's task are much lower than what we have been getting in the past (only 6% over a poor baseline vs. 23-35% over reasonable baselines). We also observe that assuming the bottom ranked documents to be non-relevant gives us some additional improvement in performance; actually an important 7-8% (over not assuming non-relevance) for this year's task. Also our reranking of the top documents to select a new set of twenty documents for feedback also gives us additional improvement across tasks, *except for this year's task*.

We believe that locality based reranking of top documents to select a better set of *assumed relevant* documents is a promising way to improve the quality of pseudo-expansion, but it needs more careful investigation. Since we developed this reranking scheme in the final days before the submission, we did not study various other alternatives that can be used for reranking in place of the above method. Also, the formula used above to marginalize the less important words was developed at the last moment and we believe that there are better ways of emphasizing core query terms than the adhoc formula we have used above.

Title-Only Queries

We also submitted a run for the very short, title-only queries. Our main motivation for this run was to test the robustness of our algorithms for these very short queries (which are very common these days in a web-search type environment). Table 9 shows the effect of various components of our adhoc algorithm on retrieval using these very short queries. For this task, pseudo-feedback doesn't yield much better results over basic vector matching, but pseudo feedback with reranking does yield about 14% improvement. This indicates that document reranking (for pseudo-feedback) is quite useful even for these tiny queries. For a casual searcher precision at twenty is usually a more meaningful number than average precision. Table 9 also shows the P@20 figures. We again see that reranking based pseudo-feedback gets (on an average) about one extra relevant document in the top 20 documents as compared to basic vector matching.

Query Length

We also study the effect of using longer user queries in adhoc searching. Table 10 shows the results of using the title-only queries, title+description queries, and title+description+narrative queries for this year's adhoc task. This scenario is akin to when a user progressively fleshes-out the query by further describing

Run	Average Precision	Best	>= Median	< Median
att97ac (desc only)	0.1847	1	36	14
att97ae (desc only)	0.1801	3	33	17
att97as (title only)	0.2289	7	31	19

Table 11: Results for adhoc runs

his/her information need to a system. The query length in column 2 is the average number of unique words and phrases in the query. In adding the description section to the title-only query, a user adds almost another nine new words and phrases to a query (the average query length increases from 3 to almost 12). In further adding the narrative section, the user adds an average of another twenty-two new words and phrases to the query (average query becomes 33.6).

A casual user will seldom provide a system with such (33 word) long queries. However, the good news is that with 3 carefully chosen words, the retrieval effectiveness of a query using our reranking-based algorithm is almost as good as the retrieval effectiveness of a very long query. Here are some key observations from Tables 8 and 10:

- For this year's adhoc task, just assuming that the top few documents retrieved by the initial query are relevant and doing relevance feedback is not very useful. This technique has been quite successful in the past. This year, depending on what parts of a topic are used in the initial run, this techniques loses or gains up to 6% in average precision. In the past, this feedback method has yielded large improvements (see Table 8, TREC-3–5 rows).
- Assuming that documents ranked poorly by the initial query are non-relevant does help some consistently. An exception is this year's description-only query (see Table 8) for which this assumption helps noticeably. This might be due to the poor baseline, or due to some other reason. We haven't investigated this yet. But, in general, there is no harm in using this assumption.
- Reranking the top few documents based on query-word locality to improve our assumption of relevance is quite useful in general. Except for the description-only queries for this year's task, this technique consistently yields improvements over no reranking. Once again, using this technique is seldom hurtful.
- Even though adding the description section to the queries somewhat improves the the initial queries, post pseudo-feedback, it is not very useful. (The improvements obtained over using the title-only queries are listed in the rows labeled *Improvement over T*.)
- Even though full queries are about 9% better than the title-only queries initially, (Table 10, column 3), post reranking pseudo-feedback, the results are just 4% better than the title-only results (column 6). This result is encouraging for locality-based reranking, since the performance gap between the very short and the very long queries reduces post reranking and pseudo-feedback.

Experimental Run

Our experimental run **att97ae** was based on the following reasoning: since pseudo-feedback is usually useful, if we do another pass of pseudo-feedback assuming that the first pass query is retrieving more relevant documents in the top ranks, and is pushing down more non-relevant documents to ranks 501-1000, we should be able to improve the results further. This half-hearted attempt didn't prove beneficial. Our experimental run yields poorer results than our first run—att97ac.

Table 11 gives comparison to medians for our submissions. Based on the number of queries for which we have below median results, we believe that there is a lot of room for improvement in our adhoc algorithm.

Run	Measure	Best	>= Median	< Median	Exact	Too Many	Too Few
att97fcuf1	Utility-1	7	37	10	0	20	27
att97feuf1	Utility-1	7	40	7	4	18	25
att97fcuf2	Utility-2	6	41	6	0	16	31
att97feuf2	Utility-2	10	41	6	2	10	35
att97fcasp	ASP	7	43	4	0	9	38
att97feasp	ASP	13	44	3	1	7	39

Table 12: Results for filtering runs

	D12345	DAT-1	DAT-2	DAT-3	DAT-4
Approximate Size (GB)	5.21	3.72	4.16	3.70	3.40
Indexing Time (Elapsed Minutes)	131	103	105	105	101
Index Size (GB)	1.81	1.21	0.88	1.10	1.20

Table 13: VLC Sub-collections

4 Filtering Runs

Our filtering track participation relies heavily upon our routing algorithm. Using the algorithm shown in Table 2 on the filtering track data, we learn a filtering query. Using this filtering query, we retrospectively rank the *training* collection and find a *similarity threshold* for the filtering query that would maximize our evaluation measure (utility or average set precision) on the *training* documents. Any test document that has a similarity greater than the above filtering threshold (to the filtering query) is assumed relevant and is passed to the user (if there were any). One should note that we optimize our filtering query to maximize average precision using DFO (step 4 in Table 2), and use the same query across evaluation measures. The only difference between different evaluation measures is in learning of the filtering threshold.

Table 12 shows the performance of our runs using the pooled evaluation. Runs att97fcuf1, att97fcuf2, and att97fcasp use the (conservative) one-pass algorithm from Table 2; whereas runs att97feuf1, att97feuf2, and att97feasp use the two pass algorithm (used in our routing run att97re). In general our filtering algorithm works well. The two-pass algorithm is somewhat better than our one-pass algorithm but we suspect that the difference is not statistically significant (we haven't done the tests yet!).

Also shown in Table 12 is an evaluation of our thresholding algorithm. The last three columns show how our threshold is doing as compared to an "ideal" threshold. The *Exact* column shows the number of queries for which our threshold did as well as the ideal threshold. The *Too Many* column shows the number of queries for which we retrieved more documents than we should have (so we had a lower threshold than the ideal threshold), and the last column shows the number of queries for which we had a higher threshold value than the optimal value. It is informative to know that the same thresholding algorithm does reasonably for utility-1 (3, -2, 0, 0), whereas for utility-2 (3, -1, -1, 0) and for average set precision, we seem to be retrieving too few documents in general. We plan to investigate our thresholding strategy in the near future, and possibly develop a more informed thresholding strategy.

5 VLC

To participate in the very large collection track, we have developed a new distributed version of the SMART retrieval system. The main design principle behind this version is: given a very large collection, it could be divided into several small, independent collections, which, when searched individually yield compatible document scores for a given query.

In the indexing phase, parts of the large collection are assigned to various CPUs (or machines on a LAN) as "independent" collections. The indexing is run in parallel on various CPUs. On our machine, the SMART system indexed the VLC text at about 2.4G/Hour. This could have been faster, had we limited ourselves

Average Query Length	27.48 words
Baseline Task P@20	0.348
Full Task P@20	0.530

Table 14: VLC Results

to running at most three indexing runs at a time instead of the five that we ran (since both the source text, and the indexed collection are stored on partitions of three striped disks and running more than three I/O bound processes usually slows down each of them due to disk bottleneck). We divided the VLC corpus into the following sub-collections: D12345, DAT-1, DAT-2, DAT-3, and DAT-4. (In retrospect, removing one of the disks from TREC D12345, and distributing it over DAT-1, DAT-3, and DAT-4 would have been a better distribution.) Table 13 shows some statistics for these sub-collection. Since our documents are *Lnu* weighted, we do not need term idf values at the time of indexing the collections, therefore all collections are indexed without any dependence on one-another. The total indexing time is the same as the longest time taken to index any sub-collection.

Once all the sub-collections are indexed individually, they read the dictionary and the df statistics for all other collections (df can possibly be encoded in the dictionary itself). Each collection merges the df information from all other collections to obtain a global df value (thus the idf value) for every term. Now, each collection has the true idf for every word. For the current implementation of the SMART system and for this task, this means reading about 50-75 MB of information from four other sources. Since all disks are local on our multi-processor system, this reading and merging took less than a minute for every collection. Of course, all this is possible since the stemming algorithm and the stop-word list is common across collections, the dictionaries across collections have same stems for a given word, and are therefore compatible.

For searching, a query is sent to each collection, and each collection retrieves its top twenty documents (twenty because this was the number wanted for evaluation in the VLC track). The similarities assigned to these documents are compatible since all collections have the global idf information for a term, as well as a common stemming/stopping algorithm (we are using *ltu* weighted queries, and we are using all sections—title, description, narrative—in the query, and we don't use phrases). The five lists of twenty documents each are merged, sorted by document score, and the top twenty documents are retrieved for evaluation. The whole retrieval take about two minutes for all fifty queries on our machine. The results are shown in Table 14. The full-task precision at twenty documents is very respectable, even better than the precision at twenty for the baseline task (a much smaller 2G database).

6 Conclusions

Our routing algorithm using query zones, word-pairs, and dynamic feedback optimization seems to be doing well. One big question that we should ask ourselves is why aren't we seeing much improvement in the routing performance over the last few TRECs? Doing a assessor agreement study in the routing environment would be interesting and might also tell us more about limits on our system performance.

Different components of our adhoc algorithm work well on different adhoc tasks. Overall, all components put together do yield noticeable improvements over a straight vector-match retrieval. As the adhoc task gets harder, with many queries with very few relevant documents, performance of the various components of our adhoc algorithm becomes unstable.

We show that it is feasible to index/retrieve-from very large text collections efficiently by sub-dividing them into smaller collections and sharing the collection information. We are encouraged by the retrieval effectiveness and the speed of our algorithms for very large collections.

Our routing algorithm followed by similarity thresholding seems to be doing a reasonable job of binary documents classification (filtering). Similarity thresholding should be studied more for the filtering environment.

Acknowledgments

We are thankful to David Lewis and Mandar Mitra for the useful discussions that helped us in various aspects of this work.

References

[1] Chris Buckley and Gerard Salton. Optimization of relevance feedback weights. In Edward Fox, Peter Ingwersen, and Raya Fidel, editors, *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 351–357. Association for Computing Machinery, New York, July 1995.

[2] Chris Buckley, Amit Singhal, and Mandar Mitra. Using query zoning and correlation within SMART: TREC-5. In D. K. Harman, editor, *Proceedings of the Fifth Text REtrieval Conference (TREC-5)*, 1997 (to appear).

[3] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[4] D. K. Harman. Overview of the fourth Text REtrieval Conference (TREC-4). In D. K. Harman, editor, *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, pages 1–24. NIST Special Publication 500-236, October 1996.

[5] Marti A. Hearst. Improving full-text precision on short queries using simple constraints. In *Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*, pages 217–232, Las Vegas, NV, April 1996.

[6] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In Hans-Peter Frei, Donna Harman, Peter Schauble, and Ross Wilkinson, editors, *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29. Association for Computing Machinery, New York, August 1996.

[7] Amit Singhal, Mandar Mitra, and Chris Buckley. Learning routing queries in a query zone. In Nick Belkin, Desai Narasimhalu, and Peter Willett, editors, *Proceedings of the Twentieth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 25–32. Association for Computing Machinery, New York, July 1997.

[8] E. M. Voorhees and D. K. Harman. Overview of the fifth Text REtrieval Conference (TREC-5). In D. K. Harman, editor, *Proceedings of the Fifth Text REtrieval Conference (TREC-5)*, 1997 (to appear).