RMIT University at TREC 2004

Bodo Billerbeck   Adam Cannane   Abhijit Chattaraj   Nicholas Lester
William Webber   Hugh E. Williams   John Yiannis   Justin Zobel

School of Computer Science and Information Technology
RMIT University, GPO Box 2476V, Melbourne 3001, Australia.
{bodob,cannane,abhijit,nml,wew,hugh,jyiannis,jz}@cs.rmit.edu.au

January 31, 2005

1 Introduction

RMIT University participated in two tracks at TREC 2004: Terabyte and Genomics, both for the first time. This paper describes the techniques we applied and our experiments in both tracks, and discusses the results of the genomics track runs; the terabyte track results are unavailable at the time of manuscript submission. We also describe our new `zettair` search engine, in use for the first time at TREC.

2 The Zettair Search Engine

The `zettair` search engine is a publicly available system developed by the Search Engine Group at RMIT. It is intended to be a straightforward implementation of effective and efficient query evaluation techniques based on current information retrieval research. `Zettair` is available under a BSD license from `http://www.seg.rmit.edu.au/zettair`. This is the first year that `zettair` has been used at TREC. The latest release version (0.6.1) is capable of indexing the GOV2 collection on a single machine, assuming sufficient disk space. `Zettair` is written in C and has been ported to many different operating systems, including Linux, Microsoft Windows, Sun Solaris, FreeBSD, and Mac OS X (Darwin).

`Zettair` can extract text from SGML-based languages — although it makes no pretense at full SGML processing — with direct support for indexing TREC collections. Users can select which portions of SGML-based languages are indexed by editing a configuration file. `Zettair` implements an efficient index construction algorithm (Heinz & Zobel 2003), with the exception that in-place merging (Moffat & Bell 1995) is not currently implemented and a variable-byte compression scheme is used instead of Golomb coding (Scholer, Williams, Yiannis & Zobel 2002). Indexes contain full word positions, which allow the resolution of phrase queries in addition to simple ranked queries. The indexing algorithm used is single-pass, and generates compressed postings in-memory and writes them to disk to limit the amount of memory used during indexing. Our implementation currently allows a (rough) configurable limit on the amount of memory used during indexing. On-disk postings are merged together to keep the number of different postings runs manageable, and then a final merge creates the final index from the on-disk postings. The final merge ensures that all postings for each term are stored in a single, contiguous list, and bulk-builds a B$^+$-tree vocabulary structure.

Multiple ranking metrics are supported, including pivoted Cosine (Singhal, Buckley & Mitra 1996) and Okapi BM25 (Jones, Walker & Robertson 2000). Our submissions use the Okapi metric, with the exact ranking formula:

$$bm25(q,d) = \sum_{t \in q} \log\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \times \frac{(k_1 + 1) f_{d,t}}{K + f_{d,t}}$$

where terms t appear in query q; there are N documents in the collection; a term occurs in f_t documents and in a particular document $f_{d,t}$ times; K is $k_1((1-b)+b \times L_d/AL)$; constants k_1 and b are set to 1.2 and 0.75 respectively; L_d is the length of a particular document as measured in bytes, and AL is the average document length over the collection. In this formulation, term contributions for terms that occur in more than half the documents in the collection are negative. In this case, a small positive term contribution is used instead.

`Zettair` can read directly from TREC query files and produce ranked output that is compatible with `trec_eval`. When it has access to the original source text, `zettair` can provide query-biased summaries.

2.1 Parsing

The `zettair` document parsing system is designed to make a single pass over the source document with only minimal buffering necessary for correct parsing. Due to the frequently malformed nature of the HTML that the search engine is required to process — for TREC tracks and other tasks — the HTML parser validates each tag by ensuring that each open tag character (<) has a corresponding close tag character (>) within a configurable numbers of characters, currently set to 999. It also ensures that no un-escaped additional start tags occur within this span. HTML comments, whose content is ignored during indexing, are also validated in this way (matching <!-- to --> without additional occurrences of <!--), as are entity references (such as matching & to ;). Limited standard HTML entity references are recognised and are substituted by their entities. `Zettair` does not currently support internationalised text, so only entities within the ASCII character set are translated. Unrecognised entites or invalid constructs are treated as if they are plain text. A primary aim of this treatment by the parser is to limit the amount of material that malformed markup can cause the parser to ignore during indexing.

Once tags and other HTML entities are recognised within the source text, a configuration file is used by the parser to determine which tags to index content between. The default is to index any material that is not explicitly excluded. Our current implementation indexes all material in source documents with the exception of material contained within comments, JavaScript, and `script`, `style`, and `vbscript` tags. Material contained within TREC document header tags is also excluded from indexing.

Documents are delimited within TREC collections by the </doc> tag. This is not foolproof, both because some of the TREC collections contain spurious empty documents that consist of only a </doc> tag, and because the source documents may also contain the sequence </doc>. The result of this, and possibly other factors, is that some documents lack a document number identifier, which makes retrieving them in response to queries problematic. In the rare case that a document that does not have a document number is retrieved in response to a TREC query, the document number of the prior document (or closest prior document with a document number) in the same source file is returned instead.

Another problem that the parsing system attempts to work around is to avoid parsing binary documents. Some previous TREC collections, notably WT10g and .GOV, have contained numerous binary documents, and GOV2 is the same. The parser ignores binary documents using heuristics, and all content in those documents is ignored until the next </doc> tag. The heuristics include searching for well-known byte sequences near the start of the document, in the same manner as the `file(1)` utility on Unix-like systems. A small subset of heuristics for document types judged most likely to occur in the TREC collections, such as PDF and Microsoft Word documents, was imported from the FreeBSD implementation of the `file(1)` utility. To apply these heuristics it is necessary to find the start of a document within the byte stream. This is difficult, as TREC collections have different formats for delimiting the content and the meta-data associated with each document. The parsing system treats any construct within the set of tags with prefix "doc" at the start of the file as header information.

Text that is not removed by HTML parsing is tokenised into separate terms for indexing. Our term-breaking heuristic ends a term when any of several criteria are met: a whitespace character is encountered, a HTML tag is encountered, control or extended ASCII characters are encountered, or two successive punctuation marks are encountered. Only letters and digits are preserved within the term for indexing purposes.

3 Terabyte-track runs

3.1 Baseline

Our baseline run used the Okapi BM25 formulation described in the previous section. Given the system-to-system differences of participants, such as specifics of parsing, our expectation is that this run will allow us to identify correspondences and differences between `zettair` and other runs.

3.2 Anchortext

This run aggregated the results of a standard `zettair` evaluation, as described above, with those from a parallel anchortext index. This aggregate index was then used as the base index for the fuzzy phrase and priority compound metrics described below.

The anchortext index was constructed as follows. For each document d in the collection, the texts of all links pointing to d from other documents in the collection were concatenated into a parallel (possibly empty) anchortext document $A(d)$. All parallel anchortext documents were then written out as a document collection in TREC format, with each $A(d)$ having the same TREC document identifier as its corresponding d. This parallel collection was then indexed in the standard way to create a parallel anchortext index.

Query evaluation then involved submitting the query to both the main and the anchortext indexes, and combining the results. The main index used the default Okapi metric described previously. The anchortext index used a metric for anchortext collections (Hawking, Upstill & Craswell 2004), which we refer to as the Hawkapi metric. Taking the final number of results required as n, a total of $5n$ results were extracted from the main index, and the same number again from the anchortext index. For each main index result, if it also occurred in the anchortext results, its anchortext score multiplied by a weighting factor was then added to its main score. These results were then sorted, and the top n results returned.

Two points should be noted about the combined metric. First, a document cannot appear in the final results if it only appears in the anchortext results, regardless of its score. Second, a main result can only have its score supplemented by its anchor result if it is in the top $5n$ anchor results; anchor results that did not meet this threshold were not considered.

Several different anchor weightings were tried and their precision calculated using our own evaluations, based on a limited manual evaluation of old .GOV queries on the new .GOV2 collection. The best weighting using this method was 0.2, which was the weighting used for the final submission. Since the Okapi BM25 and Hawkapi metrics are not normalized against each other, no particular significance should be placed on the precise value of this metric.

Our sample evaluations showed the anchortext-enhanced index provided better precision than the plain index. There are several directions for future work on enhancing results with anchortext. First, a method for combining plain and anchortext results that did not require a document to appear in the top $5n$ plain results in order to make it to the final results should be considered. Second, the Okapi and Hawkapi metrics could be normalised, making the precise values used for weighting the anchortext enhancement more meaningful.

3.3 Fuzzy Phrase

We have had positive experiences with the use of phrases in past work, and so we submitted a run in which each query was modified by repeating it as a *fuzzy phrase*. A fuzzy or sloppy phrase is a phrase query in which the terms of the phrase do not have to appear precisely in sequence; there can be gaps between them, up to a certain limit, and possibly their order can also be rearranged.

Our implementation of fuzzy phrases allowed a sloppiness factor s to be specified at query-time. This sloppiness factor allows words in a fuzzy phrase to be up to s places from their correct position. The syntax used is *"term1 term2 term3" [sloppy: s]*. For example, the phrase *"A C" [sloppy: 2]* would match *A C*, *A x C*, *A x x C*, or *C A*, but not *C x A* or *A x x x C*.

The fuzzy phrase metric was implemented by taking the query, and then modifying it by appending to the end of the query a copy of the query as a phrase with a sloppiness of 5. So, for example, the query *A B C* becomes *A B C "A B C" [sloppy: 5]*. This modified query was then submitted to the base anchortext index described in the previous section.

We allowed the fuzzy phrase metric to use a different anchortext weighting from that used for the base anchortext run. Several different anchor weightings were tried for the fuzzy phrase metric and their precision calculated using our own evaluations. The best weightings were in the range 0.6 to 1.0; we chose 1.0 for our submitted run and, again, no significance should be placed upon the precise value chosen. However, we note that the anchortext weighting here is much higher than for the base scheme. We also observed in our sample evaluations that, as we expanded the pool of evaluated documents and allowed for recall to be calculated to a greater number of results, the optimal anchortext weighting decreased. The reasons for this are unclear. We will wait for the official results and relevance judgements to investigate this further.

Our sample evaluations showed the fuzzy phrase metric performing somewhat worse than the base anchortext metric, and even than the plain (non-anchortext) metric, which was a surprise to us. We await the official results to confirm this observation.

Several modifications to our fuzzy phrase metric suggest themselves. First, we would like to be able to specify the weight to give to the phrase-term as against the plain terms in the combined query. Second, it might be desirable to give a higher score for more exact phrases than for more sloppy ones.

3.4 Priority

We also experimented with preferencing or giving *priority* to documents that matched one or more criteria. The candidate criteria were: contains the query as a phrase; contains the query as a sloppy phrase; contains all query terms in the first N words of the document; and, contains all query terms in the HTML title of the document. Experiments on the .GOV collection suggested that the best combination of criteria was sloppy phrase (with a sloppiness of 5) and all query terms in the first 50 words of the document. In the actual submission, this scheme was denoted FunkyZ.

The metric was implemented as follows. Taking the final number of results required as n, we retrieve $5n$ results from the plain (non-anchortext) index. We take the score of the top result as our *boost amount*, b. Then, we check each of these result documents to see if it meets our criteria. For each criterion it meets, we add b to its score. We then supplement these results with scores from the parallel anchortext index; the anchortext weighting used was 0.2.

With this metric, any document meeting M criteria — no matter how low its base score but provided it appears in the top $5n$ results — will be ranked higher than any document meeting only $M - 1$ criteria. This is regardless of its base score.

Although it had been quite promising in the experiments on .GOV, this priority scheme performed terribly using our sample evaluations on .GOV2. Examining the actual documents returned, we observed that these documents did indeed (as expected) have the query terms at the top of the document as or nearly as a phrase. The results, however, contained a high proportion of link documents, that is, pages containing lists of hypertext links to other pages on the subject. Since the relevance judging criteria required the page itself to directly contain the information being searched for, this may have been one reason why the priority scheme performed so badly in our sample runs.

This implementation of the priority scheme was simplistic. Consideration should be given to making the boost amount smaller, so that otherwise high-ranking documents that do not meet the extra criteria can actually remain highly-ranked. The top-N (for us, top-50) criterion could be changed to take account of the document length, although not as a linear proportion of it. In evaluating the top-N criteria, we would also like to be able to strip out navigational text, although it is not clear whether this was a serious issue for the .GOV2 collection.

3.5 Query expansion

For the expansion run we used the local analysis method proposed by Robertson & Walker (1999), where 25 terms with the lowest *term selection value* are chosen from the top 10 ranked documents:

Table 1: *Effectiveness of terabyte runs*

Technique	R Precision	Average Precision	Precision at 10
Plain (ZETPLAIN)	0.2986	0.4898	0.2230
Query expansion (ZETBODOFFFF)	0.2780	0.5122	0.2189
Combined (ZETANCH)	0.2940	0.5020	0.2168
Priority (ZETFUNKYZ)	0.2854	0.4592	0.2067
Fuzzy phrase (ZETFUZZY)	0.1965	0.3653	0.1306

$$TSV_t = \left(\frac{f_t}{N}\right)^{r_t} \binom{R}{r_t}$$

where term t appears in f_t of all N documents in the collection, and locally in r_t of the top R documents. Expansion terms were added to the query, and their weight (Robertson & Walker 1999) was derived using the formula:[1]

$$\frac{1}{3} \times \log\left(\frac{(r_t + 0.5)/(R - r_t + 0.5)}{(f_t - r_t + 0.5)/(N - f_t - R + r_t + 0.5)}\right)$$

Although we have previously found that the number of top ranked documents used and terms chosen for expansion is highly collection-dependent (Billerbeck & Zobel 2004*a*), in the absence of additional collection information (such as relevance judgements of previous years) we used the default values.

We found in unpublished experiments that placing the restriction on expansion terms that they need to occur in a minimum number of ranked documents can be beneficial; for example, it can be effective to select only the terms that fulfil the condition $r_t > 2$. However, in our preliminary experiments this restriction did not seem to help with this collection. We therefore did not restrict the selection of expansion terms.

No use was made of anchor text or any other query-independent additional information for the query expansion run; documents were ranked using only their full text.

We plan to validate alternative and more efficient methods of expansion (Billerbeck & Zobel 2004*b*) using the relevance judgements for this collection.

3.6 Results

Results are shown in Table 1. We were surprised that no technique is on average significantly better than full text or full text with anchortext. For example, query expansion is the subject of a long-term research project at RMIT and is well-understood. It works well for some topics (the highest MAP for 4 topics, and above median for 34); but has not worked well on others. The fuzzy phrase method was particularly surprising, as it was the most effective on our training runs, where admittedly the number of queries was small, and in previous TREC environments.

To explore these results further, we used the relevance judgements to see if we had, for example, made poor choices of parameters. However, these experiments only yielded small improvements, and in no instance exceeded our baseline run.

We also discovered that our baseline run was less effective than similar baselines reported by other groups at TREC. This appears to be due to differences in elements such as parsing and stemming rather than the strategies used for similarity measurement.

[1]The fraction $1/3$ was recommended by the authors in unpublished correspondence.

4 Genomic-track Runs

We participated in the Genomic track for the first time and submitted two runs for the ad hoc retrieval task. Similarly to our terabyte-track experiments, our baseline run used the Okapi BM25 formulation. Given the system-to-system differences of participants, this should again allow us to compare our second run to other non-RMIT runs.

4.1 Structure of Genomic Data

In structured document search, it is important to identify what parts of the document should be searched (and which should be ignored) in response to ranked queries. The strictly-defined structure of genomic data lends itself well to the process of selecting the best possible representative elements from queries and data. This approach has been shown at TREC 2003 to be highly effective (Osborne, Cuminskey, Sinclair, Smillie, Webber, Chang, Mehra, Rotemberg & Altman 2003).

The process of selecting fields requires careful consideration. In developing our techniques, we performed exhaustive experiments to assess which combinations of fields from queries and data leads to the most effective representation. With the training queries, we found accuracy improved by almost 1% by simply ignoring 18 fields from the Medline records[2].

4.2 Query Expansion

Variation exists in how concepts are expressed in queries and documents. These variations range from simple differences such as changes in case and word-separating punctuation, to more complex differences such as the use of abbreviations, acronyms, synonyms, plurals, and tense. Variations give rise to ambiguity in the data, and typically result in false negatives.

Case-folding overcomes differences between terms by representing all terms uniformly in a single case. We case-fold in our experiments. Punctuation variations can be resolved using pattern recognition and replacement schemes; we use the `zettair` parser in its default mode as described previously. More complex techniques are required for resolving synonyms, abbreviations, and acronyms. Stemming can resolve most issues with plurals; resolving tense is more difficult. We do not consider stemming or tense in our work.

Synonym expansion has been shown to yield significant improvements in accuracy (Hull & Waldman 2003, Osborne et al. 2003). We address the problem of synonyms by attempting to unify the different terms used for species names: when a species from a hand-crafted list is found, it is supplemented with all variants. For example, a search for the term "Mus musculus" does not match a document that contains only the terms "mouse" or "mice". Using our approach, all appearances of the term "Mus musculus" — in queries and documents — are supplemented with the additional terms "mouse" and "mice".

A hand-crafted list of 66 synonyms was created to expand terms relating to species. First, a list of species terms from past queries was obtained. Second, an online dictionary was searched for each species term and a list of words related to the species was created. Further words were added to the list by selecting unique terms from the MESH field and article abstracts from Medline data. Last, the list of terms was manually inspected to create the final set of synonyms we use in our experiments[3]. Documents and the 2004 test queries were then processed to supplement each appearance of a target-term with its synonyms.

4.3 Collection Partitioning

Conventional techniques for searching text typically examine each document in a collection for possible matches with the query. Collection partitioning is aimed at reducing the search space based on existing knowledge that a subset of documents cannot be relevant to a query. For example, a query that specifies the answer must describe a human gene cannot match documents that describe other species or do not concern genes.

[2]The ignored fields can be found at: `http://www.cs.rmit.edu.au/~abhijit/ignored-fields.html`

[3]The complete list of 66 synonyms can be found at: `http://www.cs.rmit.edu.au/~abhijit/2004species-synonyms.txt`

Table 2: *Overall effectiveness of runs, on 2004 test data*

Technique	R Precision	Average Precision	Precision at 10	Precision at 20
RMITa	0.3199	0.2796	0.5120	0.4700
RMITb	0.2508	0.2059	0.4560	0.4060

To reduce the search space, a set of *partitioning terms* need to be chosen. These terms should unambiguously divide a collection into the set of documents that may be relevant and those that are not; this occurs through a query pre-processing step using a Boolean AND of partitioning terms. When such terms are known, a query that contains a partitioning term can be evaluated on a subset of the collection, and the remainder of the collection ignored. Ideally, the terms should appear frequently in a small, but significant subset of the collection.

The search process with collection partitioning requires two processes: the first step is to execute queries that contain partitioning terms on a subset collection that contains those terms; and, the second step resolves the remaining queries — those that do not contain partitioning terms — using the entire collection. We use only names of species as the basis for partitioning, and details are described next.

4.4 Experiments and Results

Training

Queries supplied in 2004 are more structured than those used in 2003. Each contains a unique identifier, a brief description of the information need (the *Query-title*), an expanded description of the information need (the *Information-need*), and information to provide context for the search task (the *Context*). Using the training queries, we found that using all fields from each query was most effective. However, in our secondary experiment, we chose to use only Query-title and Information-need, based on our expectation this combination would work best.

Our approach to term expansion is aimed at supplementing selected terms in queries and documents with synonyms. Unfortunately, despite careful design of our synonym list, only one of the five training queries was expanded at all and this lead to poor performance for that query. Despite this, we used the technique in our submission.

We use names of species as partitioning terms for partitioning the collection in our submission.

Submissions

RMITa Our primary run for this track uses default `zettair` settings, ignores 18 selected fields from each Medline record, and uses all fields from the query.

RMITb Our secondary run combines several approaches. Collection partitioning is performed on the collection, and only the Title and Information-need fields are used from the query. Species specific term expansion is applied to both the queries and the collection.

Table 2 presents accuracy results for the RMITa and RMITb runs on the 2004 test data. RMITa performed the best of the two submissions, achieving a mean average precision 7.4% better than RMITb. Based on overall results from the ad-hoc task, RMITa achieved average precision that was the same as or better than the median for 32 of the 50 queries in the test set, and scored the highest for 2 queries. For precision-at-10 documents, RMITa scored better than the median for 30 queries and achieved the highest score for 12 of the queries. In contrast, RMITb did not perform well: both the average precision and precision-at-10 values for this run obtained or exceeded the median score for only 25 of the 50 queries. For precision at 10, RMITb achieved the best result for 9 queries.

Summary

The plain text run, RMITa, performed better than the combined collection partitioning and species term expansion approach, RMITb. Partitioning approaches work well when there is sufficient knowledge or context regarding the query and documents. In the absence of such advanced knowledge, ranked retrieval techniques are more effective. Further, only 25 of the 50 queries contained species specific data: as a result, the combination of collection partitioning and species term expansion performed poorly.

5 Conclusion

RMIT participated in the Genomics and Terabyte tracks for the first time in 2004, and used the `zettair` search engine for the first time at TREC. Results for the Terabyte track are unavailable, but our Genomics track results suggest that `zettair` is an effective text retrieval engine. In contrast, our special-purpose techniques for genomic IR were less effective, and refinement continues. We eagerly await the results of our Terabyte track runs.

Acknowledgments

This work is supported by the Australian Research Council.

References

Billerbeck, B. & Zobel, J. (2004*a*), Questioning query expansion: An examination of behaviour and parameters, *in* “Proc. Australasian Database Conf.”, Vol. 27, Australian Computer Society, Inc., pp. 69–76.

Billerbeck, B. & Zobel, J. (2004*b*), Techniques for efficient query expansion, *in* “Proc. String Processing and Information Retrieval Symp.”, Springer, Padova, Italy, pp. 30–42.

Hawking, D., Upstill, T. & Craswell, N. (2004), Toward better weighting of anchors, *in* “Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval”, Sheffield, UK, pp. 512–513.

Heinz, S. & Zobel, J. (2003), “Efficient single-pass index construction for text databases”, *Jour. of the American Society for Information Science and Technology* **54**(8), 713–729.

Hull, R. & Waldman, L. (2003), Recognizing gene and protein function in MEDLINE abstracts, *in* “Proc. Text Retrieval Conference (TREC)”, NIST—National Institute of Standards and Technology, National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, pp. 93–97.

Jones, K. S., Walker, S. & Robertson, S. E. (2000), “A probabilistic model of information retrieval: development and comparative experiments. Parts 1&2”, *Information Processing & Management* **36**(6), 779–840.

Moffat, A. & Bell, T. A. H. (1995), “In situ generation of compressed inverted files”, *Journal of the American Society of Information Science* **46**(7), 537–550.

Osborne, M., Cuminskey, M., Sinclair, G., Smillie, M., Webber, B., Chang, J., Mehra, N., Rotemberg, V. & Altman, R. (2003), Edinburgh-stanford TREC-2003 genomics track, *in* “Proc. Text Retrieval Conference (TREC)”, NIST—National Institute of Standards and Technology, National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, pp. 622–630.

Robertson, S. E. & Walker, S. (1999), Okapi/Keenbow at TREC-8, *in* “Proc. Text Retrieval Conf. (TREC)”, NIST Special Publication 500-264, Gaithersburg, Maryland, pp. 151–161.

Scholer, F., Williams, H. E., Yiannis, J. & Zobel, J. (2002), Compression of inverted indexes for fast query evaluation, *in* “Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval”, Tampere, Finland, pp. 222–229.

Singhal, A., Buckley, C. & Mitra, M. (1996), Pivoted document length normalization, *in* “Proc. ACM-SIGIR Int. Conf. on Research and Development in Information Retrieval”, Zurich, Switzerland, pp. 21–29.