Towards a Practical Information Retrieval System for the Fujitsu AP1000

David Hawking and Peter Bailey
Department Of Computer Science
Australian National University

October 25, 1993

Abstract

This paper reports progress in the development of free text retrieval systems for the Fujitsu AP1000. Work is now focussed on the classical information retrieval problem, that of retrieving documents or articles relevant to a user's query.

The current version of **ftr** permits use of the AP1000's DDV options for storing text bases, resulting in significant decreases in loading times. A new graphical user interface (called **retrieve** and based on **tcl**) which provides a user-friendly mechanism for invoking the **ftr** system from remote workstations, specifying and carrying out searches, and selecting, retrieving, viewing, and storing whole entries from the text base being searched. A range of useful tools is being developed for text base administration purposes.

Initial performance results are presented, and likely future directions for the work are outlined.

1 Introduction

The last five years have been characterised by dramatic growth in the availability of textual information in electronic form. There has been a corresponding increase in the availability of products designed to assist researchers to find relevant documents from among a large collection.

The information retrieval software reported here is a further development of the **paddy** program described in [1] and [2]. **Paddy** addressed the problem of large scale pattern matching using a parallel computer and was used to investigate the relative performance of different searching algorithms. **Paddy** was oriented toward linguistic and lexicographic research and focussed on the context in which matches occurred. While retaining **paddy** capabilities, **ftr** now addresses the classical information retrieval problem of finding relevant articles, entries or documents.

Free text retrieval software developed by ANU for the Fujitsu AP1000 now comprises:

- the **retrieve** program, providing a user-friendly, graphical means of retrieving entries, intended for end-users.
- the **ftr** program provides a character-based interface to document retrieval and also provides facilities for text base administrators.
- the nucleus of a collection of utilities for manipulating text bases

2 The retrieve Interface

The interface to the **ftr** program is through a command line parser. For everyday users, such an interface does not provide the ease of use that people have come to expect with everyday computer applications. The interface **retrieve** is primarily intended to provide a user-friendly graphical interface to a subset of the functionality of **ftr**. The interface supports a number of basic functions:

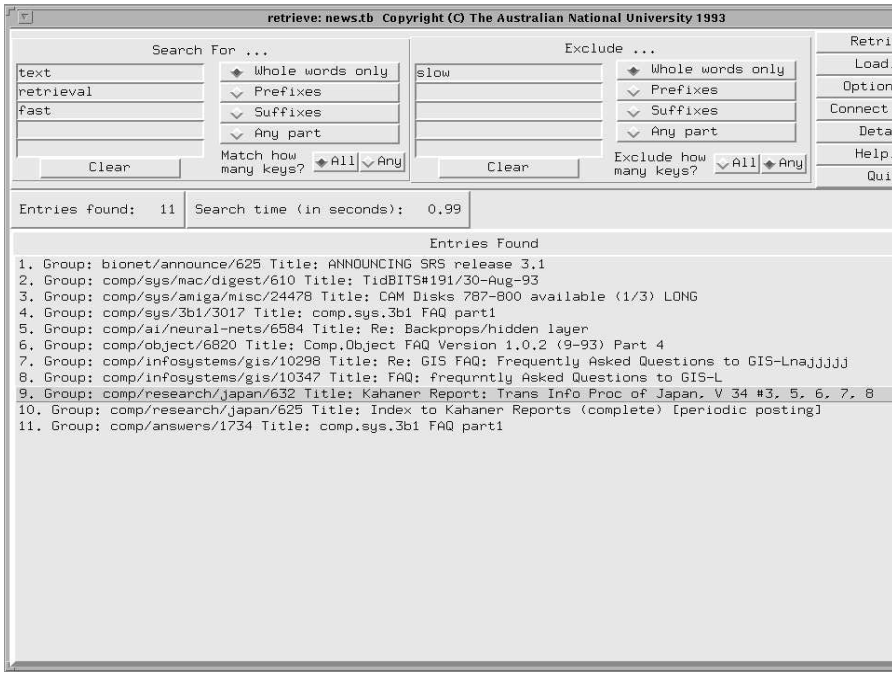

Figure 1: A screen dump of the retrieve user interface.

1. invoke a text retrieval search engine on an AP1000
2. load named text bases, in a range of storage forms
3. construct search patterns in a wide range of combinations
4. present lists of document titles matching a search operation
5. display documents selected from the lists of titles
6. save displayed documents

These features are aimed at providing what is essentially a search and browser interface for text bases. There are no abilities for the user to perform text base administration - these remain accessible only to users of **ftr**. (We hope to provide a graphical user interface for performing text base administration tasks at a later date.)

2.1 Invoking A Text Retrieval Engine

The **retrieve** interface allow the user to connect from a remote workstation to a text retrieval engine, **rftr**, that is invoked to run on an AP1000. While connected, the user can load text bases, perform searches, display and save documents. Having disconnected, the interface remains active, allowing the user to read a long article for instance (which may take much more time than connecting, loading, and searching).

2.2 Loading Text Bases

Having connected to the text retrieval engine, the user may load text bases from:

- normal text base files, stored on the host's disks;
- compressed text base files, stored on the host's disks; or

- text base files stored on an AP1000's DDV options.

Multiple text bases may be loaded. The progress in loading text bases (which can be a time-consuming task) is illustrated with a sliding scale.

2.3 Constructing Searches

The user is provided with five entry boxes for search terms (currently restricted to prefixes, suffixes and whole words) that they want to search for, and five entry boxes for terms to exclude in the search. Any of these may be left blank. The user can specify whether all of the words are required to be found in a document, or whether any of them will be adequate. Lastly, the user can specify whether terms are to be the exact word given, a word component, suffix, or prefix. Having constructed the search query, the user then presses the `Retrieve` button, and the search is invoked.

2.4 Presenting Lists Of Documents

A search will find some number of matching documents. The titles of these documents are sent back to the interface, to display in a list. The number of matching documents is reported, as is the time to search the text base. Since the sending of document titles is time consuming (there may be several hundred), a sliding scale is shown to illustrate how many have been sent.

2.5 Selecting And Displaying Documents

From the list of document titles presented, the user may scroll through the list, and double-click on a particular title. This action causes the interface to request the retrieval of the entire document so selected. The document is displayed in a separate window.

2.6 Saving Documents

Whilst reading a document, the user may decide to save it to some file for later reference. This facility is provided from a `Save` button in the display window for the document.

3 How retrieve Is Implemented

The **retrieve** interface is implemented in Tcl/Tk. The interface shares some common code with the **render** interface developed by Paul Mackerras for the CAP Visualisation project at ANU, particularly with the remote invocation of an AP1000 program engine. The interesting parts of the interface implementation are this remote invocation, and the construction of search queries.

3.1 Remote Invocation

The `Connect` button within the connection popup window invokes a `rsh` operation, which carries the appropriate directory to find the **rftr** search engine, and a socket address to connect to. On invocation, **rftr** sets up appropriate pipes to send its standard output and error back to the connection window. This facility allows the confirmation of connection. The pipes are also used by the interface to pass commands directly to the **rftr** event loop, thus driving the search engine. A socket connection is also established, directly to the Tcl event loop. The socket connection is used to pass such information as document titles, status information, and whole documents from the search engine to the interface.

3.2 Construction of Search Patterns

Once the user has constructed a particular search query using the **retrieve** interface, it must be translated into a series of set operations to be performed by the **rftr** system. The final operation is a set difference applied to the set of documents matching the include terms and the set matching the exclude terms. Each of these sets has been built up using union or intersection operators on the sets of matching documents

Collection	Total Size (Char.s)	No. of Doc.s	Longest Document (Char.s)	Shortest Document (Char.s)	*LI*
Pathological	236	1	236	236	128.00
COD8	20,696,198	40,820	26,038	1897	1.04
Awards	181,737,635	2,620	1,940,773	140,259	1.90
News	217,350,153	90,538	592,637	13,462	1.14
News2	467,519,612	167,977	12,205,045	48	3.65
OED2	545,605,058	171,476	542,524	33,627	1.02

Table 1: Document collections so far used in **ftr** and **retrieve** experiments. *LI* is the ratio of maximum to average chunk length observed after loading by **ftr** on a 128-cell AP1000.

resulting from individual searches, depending upon whether the user specified `all` or `any` in constructing the query (see Figure 1). The whole word, partial word, suffix and prefix attributes are signalled to **rftr** by placing spaces and quotation marks in the appropriate places (See [8]).

4 Large Collections Of Text Data

Information retrieval tools such as **retrieve** and **ftr** find practical application supporting research based on collections of text data such as:

- Transcripts of parliamentary proceedings;
- Government legislation;
- Law reports;
- International standards;
- Scientific papers and abstracts;
- Electronic digests, eg. computer news, financial news, etc.;
- Wirefeed news reports from eg. UPI, Reuters, AAP, etc.;
- Usenet news articles;
- Language corpera, such as the Australian Monitor Corpus and the British National Corpus;
- Electronic books, eg. Project Guttenberg;
- Library catalogues;
- Information available through World Wide Web (WWW) and Gopher.

COD8 and *OED2* are the SGML source of Oxford dictionaries and are characterised by complicated structure, rigourous and consistent markup, thoroughly edited content and a very low error rate of typographical errors. *Pathological* is an artificial case to test **ftr**'s behaviour in an extreme case. *News* and *News2* are collections of Usenet news articles characterised by unedited content and simple structure. *News2* as collected contains a 12 Mbyte document, probably a binary file, which almost certainly should have been filtered out. *Awards* is a collection of Industrial Relations data, characterised by enormous variation in document length.

It is hoped to further the development and testing of **retrieve** and **ftr** by participating in the 1994 TREC (Text Retrieval Conference) workshop. The format of the TREC workshops is described in detail

by Harman [4]. In essence, participants work with very large collections of documents including wire feed news items, electronic digests and US government publications. Initially they are given the opportunity to tune their retrieval systems using a set of training queries for which correct results are supplied. Then they must apply a new set of queries and send the results to the US National Institute of Standards and Technology (NIST).

Recall, *precision* and other results for all participants are then collated by NIST and presented at the workshop proper, at which participants have the opportunity to describe their methods and experiences.

In 1994, participating programs will be expected to handle text collections comprising over a million documents and more than two gigabytes of data.

5 Imperfections Of Real-World Document Collections

In practical applications, the text base to be searched by **ftr** will almost certainly be large (a minimum of several megabytes per AP1000 cell) otherwise it would not be economic to use an AP1000 for text retrieval. Furthermore, **ftr** will be a more useful tool if long, unbroken pieces of text are subdivided into documents of no more than a few thousand characters, such as chapters, sections, entries or even paragraphs. A researcher looking for information on a specific topic such as `zymurgy` will not be pleased if one of the documents retrieved is a 1,000 page textbook which has to be manually scanned for the one relevant paragraph!

Unfortunately, collections of documents available from external sources often lack the necessary standard or style of markup to avoid problems associated with excessive document length. Apart from inconvenience to the user, very long documents can cause imbalance in the distribution of data across cells, leading to reduced performance, inefficient memory usage and sometimes failures due to lack of memory on one or more cells. Early versions of **paddy** and **ftr** were unable to handle certain text bases in which the longest document exceeded the average chunk size. However, subject to available memory, recent improvements to **ftr** accommodate even pathological cases of imbalance.

Reliable operation over highly unbalanced text bases is ensured by the following means:

1. Searches and other operations are coded to successfully deal with the case where a cell has no data.
2. The range of shuffling operations has been extended so that a single document broken across more than two cells can be successfully merged onto one cell. See Figure 2.

The extended shuffling algorithm is as follows:

- Every cell scans its chunk for the first Start Of Document Marker (SODM)
- Every cell whose chunk contains a SODM passes the partial document preceding the SODM to its left neighbour and appends to its text chunk one or more parts of an document received from its right neighbour.
- Every cell whose chunk contains no SODM, passes its entire chunk to its left neighbour and also passes on one or more parts of the same document received from its right neighbour

In Figure 2, cell $n+1$ passes to cell n all its own data, all of cell $n+2$'s data and the tail of the article from cell $n+3$. Cells $n+1$ and $n+2$ end up with no data and cell n ends up with approximately three times the average chunk length.

The message passing protocol used in shuffling is a more complex one than the above description implies, because :

- Large partial articles must be broken into a series of smaller, acknowledged, packets to limit memory requirements of the system message area.
- Cells must be able to tell when they have received the last of a series of partial entry packets.
- The first and last cells are special cases which cannot wrap around.

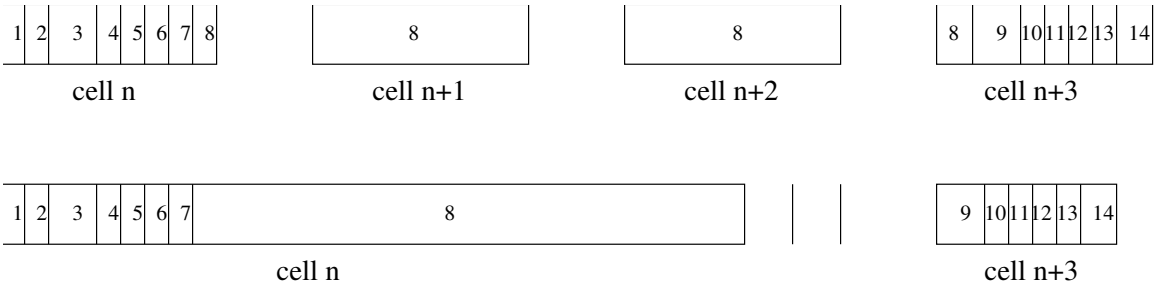

Figure 2: Shuffling a very long document. The initial distribution of data spreads document 8 across 4 cells (above). Shuffling assembles all the pieces on cell n, leaving cells $n+1$ and $n+2$ with no data at all.

The present implementation works well in cases of extreme imbalance. The artificial document collection *Pathological* consists of a single document of 236 characters. Initial loading splits it across all 128 cells but shuffling reassembles it correctly into a single document on cell 0. **Ftr** will fail if the entire collection contains less characters in total than there are processing cells but this is not considered a serious limitation! **Ftr** will also fail if a single document exceeds the memory available on a cell.

Note that shuffling applies only to uncompressed text bases loaded from the host disk. Text bases held in compressed form or on option disks have already been shuffled during initial loading.

6 Tools For Manipulating Text Bases

A practical information retrieval service is likely to rely heavily on automatic or semi-automatic software tools to manipulate, mark up, augment and prune collections of documents. The following operations have been implemented:

appendage: A sequential program for scanning a directory tree and concatenating individual text files into a single file for subsequent loading into the AP1000. Filters can be applied to eliminate subtrees (eg. "binaries" in the case of news articles) or to eliminate files with particular characteristics. Some SGML tags are generated to assist **ftr** and **retrieve**. For example the `Subject:` line in a news article may be turned into an SGML title for the document.

`merge`: A parallel **ftr** command to combine two text bases held in AP1000 memory.

`longerthan`: A parallel **ftr** command to locate all documents whose length exceeds a specified number of characters.

`purge`: A parallel **ftr** command to eliminate all documents retrieved in the last search operation.

`dump`, `compress`, `doload`, and `acload`: parallel **ftr** commands to transfer a text base in AP1000 memory to host disk, image filesystem or Acacia filesystem.

6.1 Example Use Of Text Base Manipulation Tools

An example of the practical use of the above tools would arise in the operation of a retrieval service based on Usenet news articles. The appendage program could be used to collect and markup this week's news articles into a normal sequential file, filtering out binaries and pictures and inserting document and title tags. This file could then be loaded into AP1000 memory and merged with the accumulated collection of news articles, loaded from option disks. Once merged, a search could be made for articles using `Hawking`, `text`, and `retrieval` as search terms. Articles including these terms would be purged from the text base and the resulting collection saved to the option disks.

Collection	Total Size (Char.s)	Size On Opt Disks	Load Time From Host	Load Time From Image FS	LI
COD8	20,696,198	21,615,104	12.77	4.30	1.04
News	217,350,153	248,086,528	183.20	41.24	1.14

Table 2: Loading text bases from the Image Filesystem. Times reported are the minimum of five observations.

7 Text Bases On Option Disks

In the past, the greatest limitation of the AP1000 as a text-retrieval engine has been the time taken to load text bases from disks on the front end. **Paddy** used compression techniques to reduce disk space requirements and to reduce loading time.

Now, the introduction of DDV options to the AP1000 architecture has dramatically reduced the time taken to load large data sets from disks. A proposal by Paul Mackerras of ANU for the development of FDDI connections directly to the cell B-net has the potential to similarly speed up the process of transferring external data to the cells or to the option disks.

Ftr now supports dumping and loading text bases to and from AP1000 option disks, either in the form of an animation sequence on the image filesystem or as a parallel Acacia file.

7.1 Text Bases As Animation Sequences

Advantage has been taken of the Fujitsu image filesystem's ability to seamlessly accommodate different option disk configurations (in which different numbers of cells have disk options). Of course, the image filesystem functions require each cell to dump the same amount of data, whereas the chunks of a textbase may vary considerably in size.

Each cell's chunk of text data is considered to be a series of 1,024 pixel (4,096 character) vertical scan lines. There are n vertical scan lines per frame, where n is the number of cells in the AP1000 and each cell contributes exactly one scan-line to each frame. The number of characters in a cell's chunk is padded out to a whole number of scan-lines and the number of scan-lines is padded out to the number necessary to accommodate the longest chunk of text.

Ftr text bases now include a header which records the actual size of the text chunk, so that the scan-line and frame padding can be ignored during loading.

This method works when Load Imbalance LI is close to unity, but is unsuitable for high values of LI. In the case of *News2* on a 128-cell AP1000, approximately 1.5 gigabytes of data must be stored on the option disks, though the size of the collection is less than 500 megabytes. In general, the amount of data dumped to disk is the size of the textbase (padded to a whole number of frames) multiplied by LI.

Table 2 shows load times for two sample document collections from the image file system. Load times from the option disks are very consistent compared to the times for loading via the host. The latter are subject to great fluctuation due to variations in load on the host system. The overall loading speed is approximately 5-6 million characters per second for this preliminary implementation. It is expected that this can be improved considerably.

7.2 Use Of Acacia Filesystem

The Acacia filesystem [3] now supports the AP1000 DDV options. Like the image file system, Acacia allows user programs to be written independently of the AP1000 option configuration. `Acload` and `acdump` functions have been implemented in **ftr** but, as yet, have not been tested.

Text bases are stored as a parallel Acacia file. When dumping each cell specifies the offset in the file at which writing is to occur. A cell's text chunk is padded out to a whole number of 16 kbyte blocks, but cells may dump different numbers of blocks. Consequently, there is no significant penalty when dealing

with text bases characterised by high values of LI. During loading, information kept in the text base header is used to recreate the correct file size.

8 Directions For Future Work

8.1 Improving Recall And Precision

To ensure that all relevant documents are retrieved (perfect *recall*) and that a high percentage of those retrieved are actually relevant (high *precision*) a number of improvements are proposed to the **retrieve** program:

- Use of a thesaurus to search for semantically related terms as well as specific terms specified by the user to improve recall;
- Giving access to proximity (eg. "economic" **near** "miracle") and component-limited (eg. "Bailey" **within** "authorname") searches available in **ftr** to improve precision;
- Possible implementation of *relevance feedback*, in which retrieved documents whose relevance has been judged by the user are used to automatically generate more sophisticated queries, to improve precision.

8.2 Improvements To The retrieve Interface

The current mechanisms for displaying documents and document titles are customised for an example news article text base. In future, we aim to allow the specification of appropriate document display from markup text via additions to the text base information file in the form of Tcl scripts. Such facilities will be essential for the development of document display mechanisms for searches over non-English (particularly Japanese) text bases.

The design of an easy-to-understand and easy-to-use graphical interface to support more advanced queries presents a challenge. An appropriate balance must be found in providing functionality that is adequate for the majority of operations, while avoiding becoming too cumbersome for simple ones.

Development of specialised browsers will also be considered. For instance, the interface to a library catalogue system should be quite different from that for a news article retrieval system. The specification of what aspects of the interface are common to all retrieval applications, and how to arrange the integration of plug-in modules for handling the customised aspects of the interface, is an interesting challenge.

8.3 Relevance Rankings

Ranking the relevance of documents would also improve the usefulness of **retrieve** and **ftr** programs. Retrieved documents would be listed in order of decreasing likelihood of relevance as measured by the program. The first document in the list would be the one the program judged most likely to be relevant.

A suitable measure of relevance would take into account the frequency of occurence of search terms within the document, the frequency with which the terms occur within the whole collection of documents and perhaps the contexts in which they occur.

Methods of sorting retrieved documents have been considered in the **paddy** context by Bailey, Walker and Power [7]. The work of Tridgell, Zhou and Brent [5] is also relevant. The ability to access the list DMA capabilities of the AP1000 hardware could speed up the sending of retrieved messages to the host.

8.4 Handling Free-Form Queries

Queries in the TREC workshop are expressed as *topics* which include a paragraph of English text describing the subject area of relevance. It would be desirable to incorporate into **retrieve** a content analysis module to generate from such a topic description a list of search terms related by appropriate set operations.

Content analysis might also be applied to the documents in the text base as a means of avoiding spurious matches when search terms have been used in a different sense or out-of-context. As an example of the retrieval of irrelevant documents containing all of the sought-after search terms, searching the *News* collection for "Dave", "text" and "retrieve" retrieves many news articles posted by people called Dave which have no relationship at all to text retrieval software, the terms "text" and "retrieve" being used in unrelated contexts.

8.5 Expanding Limits

The current version of **ftr** is able to reserve about 12.5 megabytes of memory per cell for textbases, indexes, entry tables and the results of searches. (The remaining 3.5 Mbytes is taken up by kernel, cell program, stack, other tasks and system message area.) Even if indexes are not built, there is thus a practical limit of about 10 megabytes per cell available for text data (about 1.3 gigabytes total on a 128-cell AP1000).

We intend to investigate alternative implementation models for increasing this limit, either by compressing the text and implementing searches over compressed text (see Zobel et al. [6]), or by keeping the text on option disks and storing only indexes in cell memory. Both of these approaches may reduce the flexibility of searches.

8.6 Improving Performance

The time taken by a cell to perform a brute-force search on its chunk of the text base is roughly proportional to the size of the chunk. Further, the time to search the entire text base is the time taken by the slowest cell to search its chunk. Consequently, the minimum overall search time will be achieved when all chunks are of identical size.

However, in order to simplify search and retrieval operations, **ftr** assumes that no article will be broken across cells. When loaded from host disk, the text base is initially chopped into equal sized pieces, regardless of article boundaries, but the cells engage in a process of *shuffling* to join partial articles together on a single cell. Consequently the goal of perfect load balance between cells cannot, in general, be achieved.

Load imbalance LI is defined as the ratio of maximum chunk length to average chunk length. Search times will be optimally short when LI is as close as possible to 1.0. Values of LI for various text bases after loading and shuffling on a 128-cell AP1000 are shown in Table 1.

Work on improving performance is currently focussed on balancing load between cells. Balancing a loaded text base can potentially reduce search times by a factor of LI.

Acknowledgements

Our thanks to the Australian National Dictionary Centre; Mike Greenhalgh, of the Art History Department, ANU; Tony Barry, of the ANU Library; Paul Thistlewaite, of CISR, ANU; and Richard Jones, of Computer Power, for continuing suggestions and assistance in locating and accessing large collections of text suitable for experimentation. We also wish to record our enduring gratitude to the late Harriet Michell whose encouragement and practical support was critical to the early stages of this project. Thanks too, to Robin Stanton for his continuing support and advice. Finally, we are indebted to Paul Mackerras for the graphical interface on which **retrieve** is based.

References

[1] Hawking, D.A. "High Speed Search of Large Text Bases On the Fujitsu Cellular Array Processor," *Proceedings of the Fourth Australian Supercomputing Conference* pp. 83-90. Gold Coast, Australia, (Dec 1991).

[2] Hawking, D.A. "PADDY's Progress (Further Experiments in Free-Text Retrieval on the AP1000)," in *Proceedings of the First Annual Users' Meeting of Fujitsu Parallel Computing Research Facilities* paper ANU-8, Kawasaki, Japan, (Nov 1992).

[3] Lautenbach, B.F and Broom, B.M. "A Parallel Filesystem For The AP1000," in *Proceedings of the First Annual Users' Meeting of Fujitsu Parallel Computing Research Facilities* paper ANU-9, Kawasaki, Japan, (Nov 1992).

[4] Harman, D.K. "Overview of the First Text REtrieval Conference (TREC-1)," in *The First Text REtrieval Conference (TREC-1)* US Department of Commerce, NIST Special Publication 500-207, (Mar 1993).

[5] Tridgell, A., Zhou, B. and Brent, R.P. "Efficient Implementation Of Sorting Algorithms On Asynchronous MIMD Machines," Computer Science, ANU Technical Report, TR-CS-93-06, (1993).

[6] Kent, A., Moffit, A., Sacks-Davis, R., Wilkinson, R. and Zobel, J. "Compression, Fast Indexing, and Structured Queries on a Gigabyte of Text," in *The First Text REtrieval Conference (TREC-1)* US Department of Commerce, NIST Special Publication 500-207, pp 229-244 (Mar 1993).

[7] Bailey, P., Power, R. and Walker, R. "The Development, Implementation, and Analysis of Sorting Algorithms for the Dictionary Search Program PADDY, on the Fujitsu AP1000," Honours Project Report, Computer Science Department, ANU, (Oct 1991).

[8] Bailey, P. and Hawking, D.A. *PADDY User Manual*, Department of Computer Science, Australian National University, Canberra, Australia, (Oct 1992).