Document Structure with IRTools

Gregory B. Newby[*]
Arctic Region Supercomputing Center
University of Alaska Fairbanks

Abstract

The IRTools software toolkit was modified for 2003 to utilize a MySQL database for the inverted index. Indexing was for each occurrence of each term in the collection, with HTML structure, location offset, paragraph, and subdocument weight considered. This structure enables some more sophisticated queries than a "bag of words" approach. *Post hoc* results from the TREC 2002 Named Page Web task are presented, in which a staged fall through approach to topic processing yielded good results, with exact precision of 0.49. The paper also provides an overview of IRTools and its interactive interface, as well as an invitation for IR researchers to get involved with the GridIR standards formation process.

Introduction

This year, the IRTools software toolkit was not quite ready in time for the TREC 2003 Web submission. Instead, this paper describes a set of runs on the 2002 Named Page Web track completed in October and November 2003. The paper should be interesting to TREC participants because it describes a rather different, and considerably more flexible, approach to information retrieval (IR) than described in the author's prior TREC entries (Newby, 2002).

IRTools is a software toolkit intended for IR research. Development was partially funded by the NSF, and the software is freely downloadable at http://sourceforge.net/projects/irtools. The goal of IRTools, scheduled for official release in 2004, is to operate as a programmer's toolkit for IR experimentation. It encompasses several major IR models (the vector space model or VSM, Boolean retrieval, and variations on latent semantic indexing or LSI). It enables both interactive use via a Web-based front end, and batch-oriented retrieval for TREC-like experiments.

IRTools is one of several systems being adopted as a reference system for Grid Information Retrieval (GIR, see http://www.gridir.org), a working group under the Global Grid Forum (http://www.gridforum.org), which the author co-chairs. GIR-WG has presented requirements and architecture documents (Gamiel et al. 2003; Nassar et al. 2003), and members of the working group are developing reference implementation systems as both proof-of-concept and early models for operational systems. GridIR is similar to WAIS (Kahle et al., 1992), in that multiple retrieval collections are federated in *ad hoc* ways to provide merged results. GridIR operates in standards-based environment such as Web services (http://www.w3c.org/2002/ws) and the Open Grid Services Architecture and other

[*] 910 Yukon Drive, Fairbanks AK 99775. newby@arsc.edu or http://www.arsc.edu/~newby . The research described here was partially funded by National Science Foundation grant #0352029.

Grid standards (Foster, 2003). These standards offer infrastructure for end-to-end security, event notification, and other capabilities.

In this paper, some of the back-end of IRTools is described. *Post hoc* results from the 2002 Named Page Web track results are presented. Future research is described.

Data Structure and Back End

Background

Similarly to most long-time TREC participants, the author has gone through many different variations in the code base for IRTools and predecessor systems. Fundamentally, though, these IR systems have several main components and purposes in common:

1. Document metadata, in which documents are assigned document ID numbers (docids). How many terms per document? What TREC document number (docnum), URL or other label is associated with a document?
2. Term metadata, in which terms are assigned term ID numbers (termids). How frequently does the term occur in a collection?
3. Inverted index, in which lists of docids for each termid are stored for quick lookup. How frequently does term i occur in document j, and at what locations in the document?
4. Sequential index, in which representations of documents are stored for relevance feedback, query expansion, context extracts, etc. What terms occur near term i in document j?

One of the largest fundamental technical challenges for nearly all IR systems is to quickly determine a set of candidate docids, given a list of termids as query. Set building occurs when the individual lists of docids from inverted index entries are merged (or sorted and merged). Once the sets are built, ranking of results can occur. This general approach is taken regardless of whether a Boolean AND or a Boolean OR is used, as well as for relevance feedback or other forms of query expansion.

We can consider the problem of information retrieval in terms of matrices of term-document relations. Table 1 shows a small set of documents and their term frequencies:

Table 1: Term by Document Matrix

	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
Term 1	2	0	1	0	1
Term 2	0	1	0	0	2
Term 3	0	3	1	0	2
Term 4	0	0	0	3	0

In Table 1, most terms do not occur in most documents, and most documents do not have most terms. This results in many cells with zero entries. Such a sparse matrix may be more efficiently represented as a list of postings in an inverted file, as shown in Table 2.

Table 2: Postings in an Inverted Index

Term 1	Doc 1=2	Doc3=1	Doc 5=1
Term 2	Doc 2=1	Doc 5=2	
Term 3	Doc 2=3	Doc 5=20	
Term 4	Doc 2=3	Doc 3=1	Doc 5=2

The advantage of the method shown in Table 2 over Table 1 is that significant space savings results by not storing the zero cells (well over 99% of cells in large IR test collections). Furthermore, multi-way sort and merge algorithms (see Knuth, 1998) enable stepping through the list of postings for each query term without requiring that the entire inverted index, or even a complete row of postings, be in main memory.

The benefit of the inverted index is not without a price, however: in Table 1, it is a simple matter to see what terms occur in a particular document by reading down the columns, and document statistics such as average term frequency are easily computed on the fly. With an inverted index, the other structures mentioned earlier (or something similar) are required for computing term and document weights and for query expansion.

In practice, of course, there is considerable variety in exactly what is needed by a particular IR system for effective ranking. By post-processing the inverted index, for example, it might be possible to rank entries by document weight, such that early entries are more likely to be associated with highly ranked documents.

Postings in IRTools

In past years, IRTools and earlier systems have used a variety of file structures to store the inverted index and other data about an IR test collection. The primary desire left unfulfilled by these file structures is to consider document qualities beyond the “bag of words” level. The bag of words, which is one of the fundamental (often implicit) approaches used in IR literature, looks at term occurrences in documents but not at where those terms occur. Furthermore, the bag of words model does not take document structure into account – for example, HTML documents have title tags, meta tags, paragraph tags, and so forth which might be important for computing the weight of a term in a document. Moreover, term position within documents is the fundamental element for phrase matching, or adjacency/nearness measures. Alternate structures, such as PAT arrays (Gonnet et al., 1992), may be employed for this, but for current purposes we would like to see whether the inverted index might be modified to add these capabilities.

By taking document structure and term position into account, new types of queries are enabled. “Term 1, near term 2, both in a TITLE tag.” “Term 1 and Term 2 in the same paragraph tag will be weighted twice as much as when they are not in the same paragraph.” “Term 1 and Term 2 in the same document, but without Term 3 as a table heading.”

Two challenges were encountered in implementing this level of analysis. First, the model needed to change from a bag of words, in which a posting in the inverted index is made for each term in each document, to a model where information needs to be stored for each *occurrence* of a term in each document. Secondly, in addition to fast search methods at the term level (i.e., the rows in Table 2 above), fast search on other qualities are also required, such as on the paragraph, subdocument, and offset location in a document. These goals seemed to fit well with what database management systems are good for, so MySQL was chosen for the TREC 2003 implementation of the inverted index.

MySQL, like PostgreSQL, is free and open source, and therefore suitable for use with IRTools. Both have similar capabilities and characteristics, but the availability of a C++ API for MySQL was a deciding factor for its choice. MySQL's MyISAM or INNODB table styles utilize either the Berkeley DB or similar approaches to storage on disk, in B-trees and related file structures. (We note here that IRTools has utilized Berkeley DB tables directly through their C++ API for several years.) Table 3 shows the table structure for the inverted index. The term and document data remained in Berkeley DB tables managed by IRTools directly, and will not be further elaborated on here.

Table 3: Inverted Index Table Structure in MySQL

Name	DocID	Offset	TermID	TagListID	WhichPara	WeightInSubdoc
Type	uint	usmall	uint	usmall	utiny	ufloat

The size and range of unsigned integers (uints) is 4 bytes, from 0 to 4GB, unsigned smalls (usmall) is 2 bytes (0 to 64K), and tiny integers (utiny) from 0 to 255. This nets 17 bytes per posting – that is, per term occurrence in a document, plus overhead and indexing. As shown in Table 4, an index was built on each of these database columns as well as combinations of columns, which more than doubled both insertion time and the database size on disk, but allowed many queries to run without requiring linear searches through the postings.

In the postings, Offset is simply the word number in the document, with any term offset over 64K being skipped. WhichPara is simply the paragraph number (with some simple rules for "what is a paragraph" in HTML, implemented in the LibWWW parser), with any paragraphs over 255 being mapped to 255. WeightInSubdoc is simply a traditional document weight that is incremented for additional occurrences. The ability to uniquely weigh a term occurrence within a document is powerful, but further research is needed to determine what sort of weighting scores to implement.

TagListID is the most interesting column. In the collection, a list of all HTML tag sequences was kept. For example, well-formed HTML should start (after a DOCTYPE) with an HTML tag, a HEAD tag, and perhaps a TITLE or META tag. So, the first title term might occur in a tag sequence such as: HTML, HEAD, TITLE. In the inverted index, a unique TagListID was assigned to each tag sequence. By normalizing the HTML data and forcing them to be indexed as well formed, this enables searches for terms in title tags, as well as much more specific searches (e.g., terms in italics, in table columns, in table rows, in the body section of an HTML document). In practice, it was found that just under 64K unique TagListIDs were needed for the Web02 collection.

Table 4: Inverted Index Table Creation

```
CREATE TABLE `inv0web02` (
  `docid` int(10) unsigned NOT NULL default '0',
  `offset` smallint(5) unsigned NOT NULL default '0',
  `termid` int(10) unsigned NOT NULL default '0',
  `taglistid` smallint(5) unsigned NOT NULL default '0',
  `whichpara` tinyint(3) unsigned NOT NULL default '0',
  `weight_in_subdoc` float unsigned NOT NULL default '0',
  PRIMARY KEY  (`docid`,`offset`),
   KEY `termid_index` (`termid`),
  KEY `whichpara_index` (`whichpara`),
  KEY `taglistid_index` (`taglistid`),
  KEY `weight_index` (`weight_in_subdoc`),
  KEY `docid_index` (`docid`),
  KEY `offset_index` (`offset`),
  KEY `termid_docid_whichpara_offset` (`termid`,`docid`,`whichpara`,`offset`),
  KEY `termid_docid_whichpara_offset_weight`
(`termid`,`docid`,`whichpara`,`offset`,`weight_in_subdoc`),
  KEY `termid_docid_taglist_weight` (`termid`,`docid`,`taglistid`,`weight_in_subdoc`),
  KEY `termid_docid_taglistid_offset_weight`
(`termid`,`docid`,`taglistid`,`offset`,`weight_in_subdoc`),
  KEY `termid_docid` (`termid`,`docid`)
) TYPE=MyISAM
```

Indexing the Web02 Collection

IRTools was used to index the TREC Web02 test collection of 1.2M HTML documents (about 20GB). Other than the MySQL database described above, the main innovation this year was to add a complete HTML parser. LibWWW from the World Wide Web consortium was chosen. LibWWW has proven to be fast and reasonably efficient, but poorly documented and rife with memory leaks that occur in ongoing use (such as the multi-day indexing process of Web02). Term and document information was stored in Berkeley DB files, as in prior years, and the sequential index was dropped, because it can be efficiently generated by selecting all postings for a DocID from the MySQL database.

Statistics for the Web02 collection are presented in Table 5. Terms with more than 20 occurrences in a document were arbitrarily capped at 20 (although term counts continued to accrue, data concerning the 21st term occurrence and beyond were omitted). All terms were considered, without use of a stoplist, truncation or stemming.

Table 5: Web02 Build Statistics

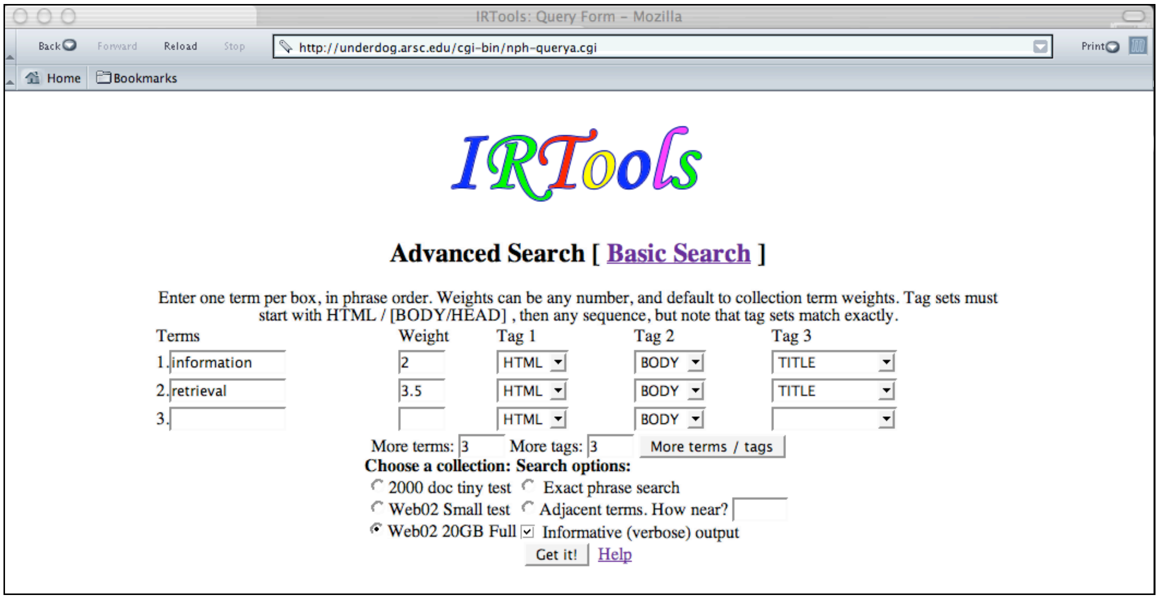
System	Dell 4600
	Dual Xeon processor 2.8Ghz
	16GB RAM
	960GB RAID-5 disk
Processing time	5 days
MySQL database size (inverted index)	About 80GB
Number of inverted rows (postings)	About 468,000,000
Size of other file structures	About 500MB
Total index size	About 81GB

We note that 468 million rows is, indeed, a large database table. At a ratio of 4:1, the size of the database compared to input data is not nearly as favorable as for other IR systems. Furthermore, the random access nature of inserts (combined with numerous indexes) resulted in insertions which were very much disk bound. While indexing, CPU utilization was often below 10%, while awaiting disk I/O. Generally this behavior is consistent with other indexes for IR, and no slower than the Berkeley DB or other B-tree disk methods. The difference was that keeping track of virtually every term occurrence resulted in a far larger database.

Query Flexibility

With the use of the MySQL database and indices on all columns, IRTools is suitable for both batch-oriented TREC topics, as well as a variety of on-demand queries. Context-sensitive document extracts are simply a matter of retrieving a range of database rows for a particular DocID. So is a title for display. Figures 1 and 2 illustrate some of the query flexibility and output options provided by IRTools through a Web-based front-end.

Figure 1: Advanced IRTools Query Form



http://underdog.arsc.edu/cgi-bin/nph-querya.cgi

Advanced Search [Basic Search]

Enter one term per box, in phrase order. Weights can be any number, and default to collection term weights. Tag sets must start with HTML / [BODY/HEAD] , then any sequence, but note that tag sets match exactly.

Terms	Weight	Tag 1	Tag 2	Tag 3
1. information	2	HTML	BODY	TITLE
2. retrieval	3.5	HTML	BODY	TITLE
3.		HTML	BODY	

More terms: 3 More tags: 3 More terms / tags

Choose a collection: Search options:

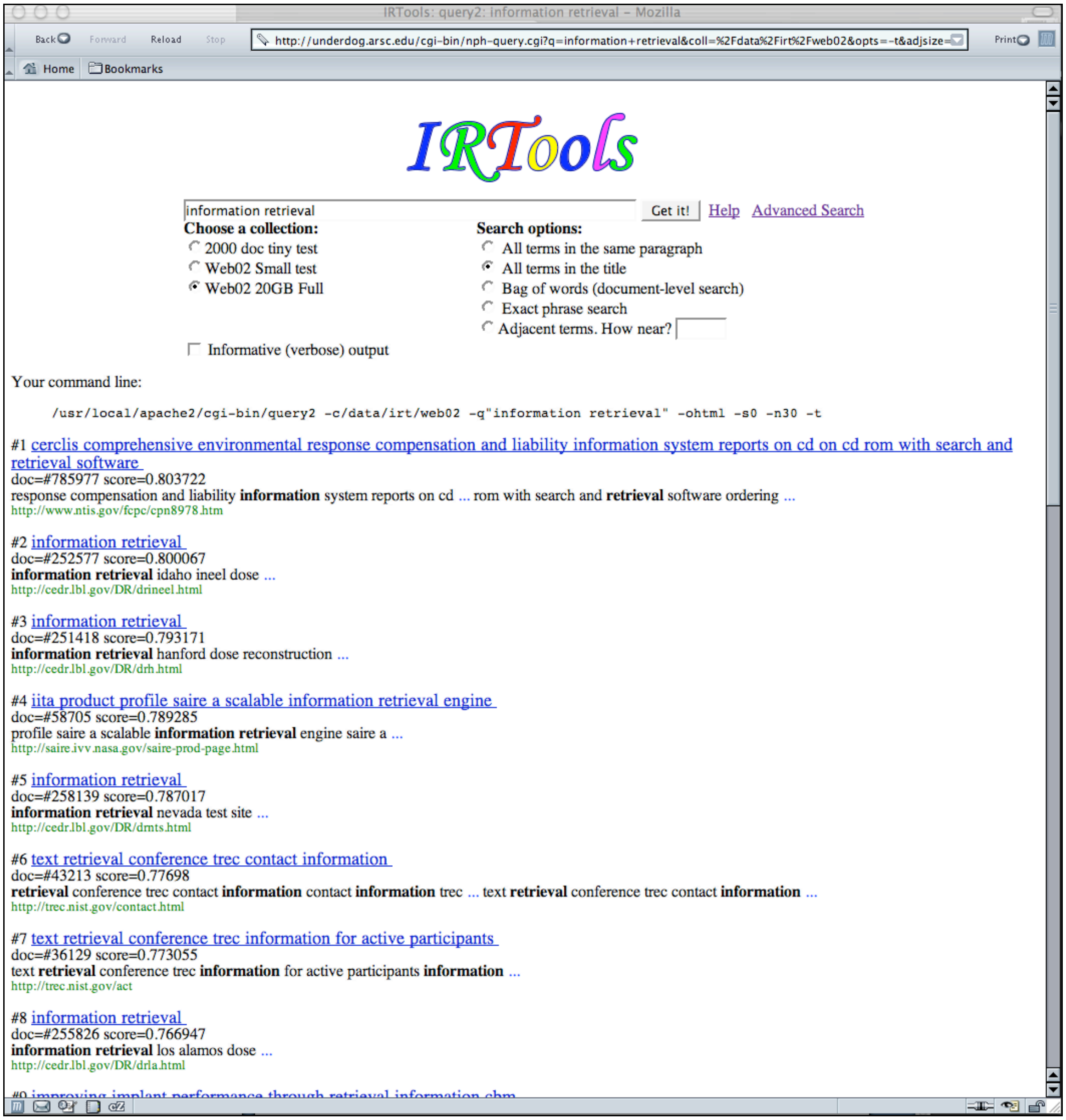
○ 2000 doc tiny test ○ Exact phrase search
○ Web02 Small test ○ Adjacent terms. How near?
● Web02 20GB Full ☑ Informative (verbose) output

Get it! Help

Tasks and Outcomes

The revisions to IRTools discussed here were not quite ready in time for the TREC 2003 Web track deadline (results from running TREC 2003 tasks on last year's system were submitted instead). Here, we present results using relevance judgments (qrels) from the TREC 2002 Named Page finding task of the Web track. In that task, the goal was to identify particular pages or Web sites based on a description of their name.

Figure 2: Basic IRTools Query with Output



http://underdog.arsc.edu/cgi-bin/nph-query.cgi?q=information+retrieval&coll=%2Fdata%2Firt%2Fweb02&opts=-t&adjsize=

information retrieval

Choose a collection:

○ 2000 doc tiny test
○ Web02 Small test
● Web02 20GB Full

Search options:

○ All terms in the same paragraph
● All terms in the title
○ Bag of words (document-level search)
○ Exact phrase search
○ Adjacent terms. How near?

□ Informative (verbose) output

Your command line:

```
     /usr/local/apache2/cgi-bin/query2 -c/data/irt/web02 -q"information retrieval" -ohtml -s0 -n30 -t
```

#1 cerclis comprehensive environmental response compensation and liability information system reports on cd on cd rom with search and retrieval software
doc=#785977 score=0.803722
response compensation and liability **information** system reports on cd ... rom with search and **retrieval** software ordering ...
http://www.ntis.gov/fcpc/cpn8978.htm

#2 information retrieval
doc=#252577 score=0.800067
information retrieval idaho ineel dose ...
http://cedr.lbl.gov/DR/drineel.html

#3 information retrieval
doc=#251418 score=0.793171
information retrieval hanford dose reconstruction ...
http://cedr.lbl.gov/DR/drh.html

#4 iita product profile saire a scalable information retrieval engine
doc=#58705 score=0.789285
profile saire a scalable **information retrieval** engine saire a ...
http://saire.ivv.nasa.gov/saire-prod-page.html

#5 information retrieval
doc=#258139 score=0.787017
information retrieval nevada test site ...
http://cedr.lbl.gov/DR/drnts.html

#6 text retrieval conference trec contact information
doc=#43213 score=0.77698
retrieval conference trec contact **information** contact **information** trec ... text **retrieval** conference trec contact **information** ...
http://trec.nist.gov/contact.html

#7 text retrieval conference trec information for active participants
doc=#36129 score=0.773055
text **retrieval** conference trec **information** for active participants **information** ...
http://trec.nist.gov/act

#8 information retrieval
doc=#255826 score=0.766947
information retrieval los alamos dose ...
http://cedr.lbl.gov/DR/drla.html

#9 improving implant performance through retrieval information chm

By its nature, this is a task that desires relatively small response sets for high precision. Only a few relevant pages were identified for most topics. Four runs, plus a combined run, were evaluated in two different scenarios.

- Run 1 searched only the HTML title tag for query terms. Weighting for this and all other runs favored the exact query phrase, but was otherwise Lnu.Ltc using the VSM.
- Run 2 looked for the exact query phrase within the same paragraph (where paragraph is defined as any block-level set of text, including tags such as p, table, and ul).
- Run 3 ranked documents containing all query terms with offsets plus or minus 10 from one another. (The actual implementation was that each query term had to be within 10 terms from the first query term.)
- Run 4 was a "bag of words" approach, in which any document with all query terms was considered for ranking.
- The fifth set was the combination of all prior sets. These results are summarized in Tables 6 and 7.

It was speculated that the four runs could operate in a "fall through" manner: if run 1 yielded no results for a particular topic, run 2 would ensue. Similarly, run 3 would only occur for a topic if run 2 yielded no results. The bag of words approach in run 4 was, essentially, a move of desperation. Thus, the "Combined" column of Table 6 is the result of all 150 topics in which one or more of the runs were completed, only the last of which produced results. This is the fall through scenario.

The other scenario is to simply use each method alone on all 150 topics. At the outset, we presumed that Run 1 would provide the highest precision, while Run 4 would find additional relevant documents but at the expense of far lower recall.

Table 6: Web02 Named Page Task Results Summary for Fall Through Queries

Recall	Precision				
	Run 1	Run 2	Run 3	Run 4	Combined
		(If Run 1 fails)	(If Run 2 fails)	(If Run 3 fails)	(Complete set)
0	0.5392	0.2461	0.1447	0.4048	0.2839
0.1	0.5392	0.2461	0.1447	0.4048	0.2839
0.2	0.5392	0.2461	0.1447	0.4048	0.2839
0.3	0.5392	0.2461	0.1447	0.4048	0.2839
0.4	0.5392	0.2461	0.1447	0.4048	0.2839
0.5	0.5392	0.2461	0.1447	0.4048	0.2839
0.6	0.451	0.2333	0.1447	0.3095	0.2483
0.7	0.451	0.2333	0.1447	0.3095	0.2483
0.8	0.451	0.2333	0.1447	0.3095	0.2483
0.9	0.451	0.2333	0.1447	0.3095	0.2483
1	0.451	0.2333	0.1447	0.3095	0.2483
# of Queries	34	13	77	21	145
Retrieved	142	104	541	75	862
Relevant	39	17	82	27	165
Relevant retrieved	20	6	22	11	59
Exact precision	0.49	0.23	0.1	0.33	0.24

Table 6 shows that relatively high precision was achieved in Run 1 (title only), but at the expense of many queries for which no results were submitted. 34 topics resulted in 142 documents identified as potentially relevant, 20 of which actually were. Most topics only produced a few documents, with only 64 ("work/life center map"), 88 ("export import bank") and 137 ("endangered species picture book") in the double digits with 31, 31, and 14 documents each, but none was relevant. Those topics are in contrast to topics that hit exactly, with one to three documents found, all of which were relevant.

Of the 116 (150 – 34) topics submitted to Run 2 (exact phrase), only 13 resulted in documents being presented, and with lower scores overall than Run 1. As would be expected, exact phrase is not necessarily indicative of a named page – and in this case, the named pages with exact phrase were more likely to be title pages.

Another 77 of the remaining 103 topics resulted in "hits" for Run 3, the adjacency search. Numerically, this was the richest set, with 22 new relevant documents found out of 82 possible for those topics. But the 519 non-relevant documents resulted in low overall scores, and a very low exact precision. The low variety in precision scores across all runs are because most topics had very small response sets, due to the specificity of the IRTools query.

Only 26 topics remained for Run 4, and 21 of these resulted in some documents being found. Failure to produce any hits on the other 5 is mostly attributable to parsing issues (i.e., "e-coli" versus "ecoli", and the infamous "u.s." versus "us" versus "u.s"). The bottom line here is that a fall through approach seems reasonable: start with more specific topics, and then try less specific queries before giving up. Adjusting the adjacency search to a smaller window, or requiring term ordering as in the topic, could help.

For comparison, runs of all 150 topics were made with each of Run 1 through Run 4. Results for Run 1, of course, matched those for the fall through method. For Runs 2, 3, and 4, results dropped off rapidly due to increasing numbers of non-relevant documents being added to the response set. Table 7 summarizes these results.

Table 7: Web02 Named Page Task Results Summary for Independent Runs

	Run 2	Run 3	Run 4
# of Queries	150	150	150
Queries with results	33	123	145
Retrieved	403	1548	1593
Relevant	38	137	165
Relevant retrieved	17	50	47
Exact precision	0.21	0.1	0.1

We see in Table 7 that the high specificity of Run 2, with an exact phrase search, did not fare much worse than in the fall through case, where Run 2 only occurred if Run 1 failed. But for Runs 3 and 4, a disproportionately large number of candidates were submitted for a relatively small number of relevant documents. These boosted the Relevant Retrieved scores, but were not especially successful methods otherwise.

Future Directions

IRTools has many facets. It is hoped that other information scientists will consider utilizing parts of it for their own research, perhaps even contributing new components. For the immediate future, the main research topic of interest is how to allocate term weights in subdocuments. Must these be computed after the initial indexing, when term characteristics for the whole collection are known? What about incorporating collection-level statistics such as page rank? What adjustments to traditional tf, idf and pivot scores are needed?

From a development perspective, the biggest effort will go to a reference implementation for GridIR. TREC participants are urged to get involved with GridIR. Apart from being of inherent interest to many of us, GridIR is a ripe platform for experimentation on query results merging, information filtering, and different document types. More information, including standards documents, is online at www.gridir.org.

References

Foster, Ian. 2003. "The Grid: Computing without bounds." Scientific American April. Online: www.sciam.com

Gamiel, Kevin; Karimi, Sousan; Newby, Gregory; Nassar, Nassib. 2003. "Grid information retrieval requirements." Online: www.gridir.org/wg_docs.html

Gonnet, Gaston H.; Baeza-Yates, Ricardo; Snider, Tim. 1992. "New indices for text: PAT trees and PAT arrays." In: Information Retrieval: Data Structures and Algorithms. Upper Saddle River, New Jersey: Prentice-Hall.

Kahle, B.; Morris, H.; Davis, F.; Tiene, K.; Hart, C.; Palmer, R. 1992. "Wide Area Information Servers: An executive information system for unstructured files." Electronic Networking: Research, Applications and Policy 1(2): 59-68.

Knuth, Donald. 1998. The Art of Computer Programming Vol. 3: Sorting and Searching. New York: Addison-Wesley.

Nassar, Nassib; Newby, Gregory; Gamiel, Kevin; Dovey, Matthew; Morris, Jeremiah. 2003. "Grid information retrieval architecture." Online: www.gridir.org/wg_docs.html

Newby, Gregory B. 2002. "Progress in General-Purpose IR Software." In Voorhees, Ellen (Ed.). TREC 2002 Proceedings. Gaithersburg, Maryland: NIST.