Implementing Information Retrieval Algorithms Using Mumps

Kevin C. Ó Kane
Professor Emeritus
Computer Science Department
University of Northern Iowa
Cedar Falls, IA 50613

https://www.cs.uni.edu/~okane/
https://threadsafebooks.com/

The author may be contacted at:

kc.okane@gmail.com

Graphics and production design by the

Threadsafe Publishing & Railway Maintenance Co.
Hyannis, Nebraska

May 23, 2022

Table of Contents

Index of Tables

Index of Programs

1 Introduction

1.1 What is Information Retrieval?

Information retrieval (IR), sometimes revered to as *information storage and retrieval*, was defined by Gerard Salton, one of the pioneers in the area, as:

> *... a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information.* [Salton 1968]

Information retrieval systems are generally composed of two parts:

1. An indexing engine that analyses digitized collections of information, often text but also images, videos, audio, scientific and clinical data, for the purposes of:
    (a) identifying, extracting, and organizing tokens that are indicative of the content, and
    (b) organizing and storing these tokens in a structured, searchable knowledge database.
2. An interactive search engine that is used to:
    (a) construct and formulate queries in terms of the knowledge structures built by the indexing engine,
    (b) execute these queries, and,
    (c) meaningfully report, rank, visualize and interpret the results

Users interact with IR systems in many ways. These range from simple text queries to a wide variety of complex graphical interfaces that facilitate navigation, visualization and interpretation of retrieved content.

In an IR system, queries are matched with objects in the database in a variety of ways from exact, Boolean matching to partial matching based on probabilistic analysis of the similarities between queries and objects.

Thus, for example, a query to a text based IR system working on a collection of medical text that seeks information on the broad search key *antibiotics* will identify many documents based on an exact match of the key *antibiotic*. Some of these documents will deal with antibiotic usage while many others will deal with related topics such as research and development, side effects and so on.

In an exact match system, only those documents containing the search key will be retrieved. Articles not explicitly containing the key will not be retrieved.

However, there may be many items that may not actually contain the search term itself but are related to the topic by semantically similar terms such as *infection control, antiseptic, antibacterial, germicidal* or as specific antibiotic agents such as *penicillin, amoxicillin, ampicillin*, *erythromycin* and so on.

While some systems will retrieve only those documents containing exactly the search term, others will enhance the results by including, at various weights, additional, related query terms.

Results will normally be presented based on a ranking to indicate which documents the IR system's algorithms determine most likely best satisfied the user's query. Rankings may be based on several factors but the frequency of usage in a document of the search key, or keys, relative to the size of the document, is one popular metric

While there are indexing and retrieval systems for other areas such as video, images, audio and so forth, in this text, we will concentrate on those basic algorithms and procedures associated with text indexing and retrieval.

2 Basic Indexing Concepts

2.1 Text Indexing and Retrieval

Perhaps the most common form of information retrieval familiar to most people involves text queries. For most of us, this means a typed query to Yahoo, Bing, Google, DuckDuckGo, Baidu, AOL, Ask, Excite and other similar Internet based search services.

In these web search engines, the user phrases his or her query as a sequence of words. Most queries, however, are just a sequence of one or more words. In some cases, however, queries are in the form of natural language sentences.

A basic query might be consist of a single overly broad term such as *aviation.* Typically, the results returned would include many references. These would include articles on early pioneers in the field, technical reports on aircraft design, flight schedules on airlines, information on airports and so on. For example, the term *aviation* when typed into Google resulted in about 111,000,000 hits, all of which presumably had something to do with aviation.

Search engines usually present results in a ranked or ordered arrangement as determined by algorithms that estimate relevancy of each result to the query. While these algorithms are normally proprietary and take into account many factors including user search history.

When confronted with poor search results, the typical user usually rephrases the query to include more specific terminology so as to narrow the search and obtain more relevant results. Most web search engines provide limited support at query reformulation as their primary interest is to display advertisements. The more user interaction, the more ads the user is exposed to.

Many IR systems incorporate knowledge of the user into account. This can take the form of user feedback, user query history or other profiles. This information helps the IR system to more closely attenuate the results to the individual.

For example, in a system that incorporates user profiles, articles retrieved in response to a query from someone whose profile is that of a grade school student will be significantly different than those returned for a someone whose profile is that of a graduate student.

In summary, a well designed IR system should involve many levels of user feedback in which the system interacts with the user. A well designed system will learn from users and adapt accordingly. A well designed IR system should also provide assistance in query formulation and database navigation.

2.2 Document Indexing

Information retrieval is the matching a user query with one or more documents in a database. Unlike a relational database system, the match is, in most cases, approximate. That is, some retrieved items will be more closely related to the query while others will be more distantly related. Results are usually presented from most relevant to least relevant with a cutoff beyond which documents are not shown. For example, the query *information retrieval* to Google resulted in nearly 14 million hits. However, only a sample of these, the most relevant, as determined by Google's relevancy algorithm, are actually displayed.

In it's simplest form, an information retrieval system consists of a collection of documents and one or more procedures to calculate the similarity between queries and the documents.

Determining the similarity between queries and documents is usually not done directly. Instead, both queries and documents are mapped into an internal representation or indexing model upon which the similarity functions can be directly calculated as diagrammed in Figure 1. The results from a query are determined by calculating the similarity between the internal representation of query and the internal representation of the documents in the context of the indexing model. Basically, this mean that queries become small documents.

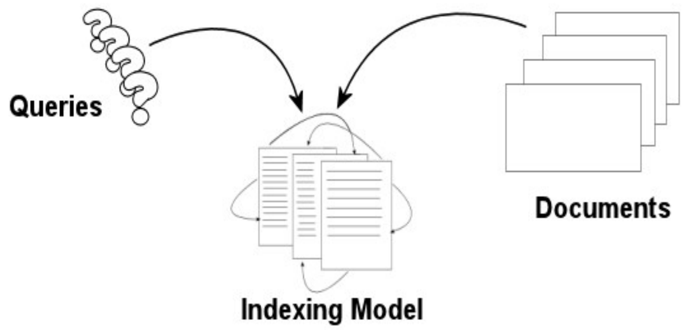


Figure 1 Overview of Indexing & Retrieval

In this model, documents are preprocessed into the internal representation, a process called indexing. This can be a very time consuming procedure when the documents in the collection may number in the millions. Once the main collection has been indexed, however, newer documents may be added more quickly.

When a query is received, it too is indexed but the cost is nominal.

3 Indexing Vocabularies

Traditionally, indexing was performed by experts in a subject area who read each document and classified it according to content. However, in recent years, manual indexing has been overtaken by automated indexing, of the kind performed by search engines such as Bing, Google and other online retrieval systems.

In any indexing scheme, there is a distinction between a *controlled* and *uncontrolled* vocabulary scheme. A *controlled vocabulary* indexing scheme is one in which previously agreed upon standardized terms, categories and hierarchies are employed. On the other hand, an *uncontrolled vocabulary* indexing system is one that derives the terms, categories and hierarchies directly from the text.

3.1 Controlled Vocabularies

In a controlled vocabulary system, topics are described using the same preferred term or terms each time and place they are indexed, thus ensuring uniformity across user populations and making it easier to find all information about a specific topic during a search. Many controlled vocabularies exist in many specific fields. These take the form of dictionaries, hierarchies, and thesauri which structure the content of the underlying discipline into commonly accepted categories.

For the most part, these are constructed and maintained by government agencies (such as the National Library of Medicine in the U.S. or professional societies such as the ACM).

For example, the Association for Computing Machinery Computing Classification System (1998):

http://www.acm.org/about/class/1998/

is used to classify documents published in computing literature. This system is hierarchical. The author or reviewer of a document indexes the document under one or more categories to which the document most specifically applies and at the level in the tree that best corresponds to the generality of the document. For example, consider the extract of the ACM hierarchy shown in Figure 2.

Multiprocessing/multiprogramming/multitasking
o Mutual exclusion
o Scheduling
o Synchronization
o Threads NEW!

* D.4.2 Storage Management
  o Allocation/deallocation strategies
  o Distributed memories
  o Garbage collection NEW!
  o Main memory
  o Secondary storage
  o Segmentation [**]
  o Storage hierarchies
  o Swapping [**]
  o Virtual memory
* D.4.3 File Systems Management (E.5)
  o Access methods
  o Directory structures
  o Distributed file systems
  o File organization
  o Maintenance [**]
* D.4.4 Communications Management (C.2)
  o Buffering
  o Input/output
  o Message sending
  o Network communication
  o Terminal management [**]
* D.4.5 Reliability
  o Backup procedures
  o Checkpoint/restart
  o Fault-tolerance

Figure 2 ACM classification system

Numerous other examples abound, especially in technical disciplines where nomenclature tends to be more precise. For example:

1. MeSH (Medical Subject Headings) for medicine and related fields

   https://www.nlm.nih.gov/mesh/meshhome.html

2. International Classification of Diseases - Clinical Modification version 9 (ICD9-CM) and related codes for diagnostic and forensic medicine:

   http://icd9cm.chrisendres.com/

3. National Library of Medicine Classification Schedule for medically related works

   https://classification.nlm.nih.gov/

   Mental disorders:

   https://www.psychiatry.org/psychiatrists/practice/dsm

4. International Union for Pure and Applied Chemistry Names (IUPAC Names) for chemistry

   https://iupac.org/what-we-do/nomenclature/

5. Library of Congress Classification System for a broad classification system for all works:

   https://www.loc.gov/catdir/cpso/lcco/lcco.html

6. Structural Classification of Proteins:

   http://scop.mrc-lmb.cam.ac.uk/scop/

7. Open Directory Project:

   http://www.odp.org/homepage.php

8. For a very long list, see American Society of Indexers Thesauri Online:

   https://www.asindexing.org/

In a manually indexed collection that uses a controlled vocabulary, experts trained not only in the field but also in the vocabulary read and assign vocabulary or hierarchy codes to the documents.

Historically, because of the complexity of the terminology and the expense of conducting online searches, these systems were accessed only by trained personnel who intermediated a user's queries and translated them into the precise vocabulary of the discipline.

Prior to the advent of the Internet, online database searching was expensive and time consuming. In recent years, however, with the advent of ubiquitous Internet access and vastly cheaper computer facilities, the end user is more likely to conduct a search directly.

3.2 Uncontrolled Vocabularies

Uncontrolled or derived vocabulary systems have been around for many years. These derive their terms directly from the text. Among the earliest forms were biblical concordances such as the King James Bible Hebrew and Greek Concordance Index:

http://www.sacrednamebible.com/kjvstrongs/CONINDEX.htm

This is an alphabetically organized index which references each occurrence of each term in the text.

A more secular example can be found in the index for John Bartlett's Familiar Quotations, 10th ed. 1919:

https://www.bartleby.com/100/s0.html

Manual construction of concordances is tedious but well suited as a computer application.

Systems based on controlled vocabularies have many limitations, most prominent of these is the amount of manual effort to maintain them. While this investment of time may be justified in limited areas such as medicine, it is not practical in most fields of interest.

Computer based uncontrolled vocabularies can be constructed through statistical analysis of word usage in the collection as a whole. Building systems based on derived vocabularies is is the main theme of this book. In Chapter 13.2 on page 186 we will examine indexing the OHSU MEDLINE collection with the controlled MeSH vocabulary.

3.3 WordNet

WordNet can be found at:

http://wordnet.princeton.edu

It is:

"*... a large lexical database of English, developed under the direction of George A. Miller. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing. ...*" *(from the web site)*

One problem confronting all retrieval systems is the ambiguous nature of natural language. While in scientific disciplines there is usually a non-ambiguous, precise vocabulary, in general text, many words have different meanings depending upon context. When we use words to index documents it would be desirable, if possible, to indicate the meaning of the term.

For example, the word *base* elicits the results shown in Figure 3 from WordNet. Even for a simple term, term are many possible meanings, depending on context.

Ideally, when processing documents, terms near the term being examined could be used to disambiguate this context. For example, the sample sentences shown in Figure 3 could provide related terms that could help select the correct sense of the term.

WordNet can be automatically installed in Linux through the Synaptic Package manager.

- S: (n) base, base of operations (installation from which a military force initiates operations) "the attack wiped out our forward bases"
- S: (n) foundation, base, fundament, foot, groundwork, substructure, understructure (lowest support of a structure) "it was built on a base of solid rock"; "he stood at the foot of the tower"
- S: (n) base, bag (a place that the runner must touch before scoring) "he scrambled to get back to the bag"
- S: (n) base (the bottom or lowest part) "the base of the mountain"
- S: (n) base ((anatomy) the part of an organ nearest its point of attachment) "the base of the skull"
- S: (n) floor, base (a lower limit) "the government established a wage floor"
- S: (n) basis, base, foundation, fundament, groundwork, cornerstone (the fundamental assumptions from which something is begun or developed or calculated or explained) "the whole argument rested on a basis of conjecture"
- S: (n) base, pedestal, stand (a support or foundation) "the base of the lamp"
- S: (n) nucleotide, base (a phosphoric ester of a nucleoside; the basic structural unit of nucleic acids (DNA or RNA))
- S: (n) base, alkali (any of various water-soluble compounds capable of turning litmus blue and reacting with an acid to form a salt and water) "bases include oxides and hydroxides of metals and ammonia"
- S: (n) base (the bottom side of a geometric figure from which the altitude can be constructed) "the base of the triangle"
- S: (n) basis, base (the most important or necessary part of something) "the basis of this drink is orange juice"
- S: (n) base, radix ((numeration system) the positive integer that is equivalent to one in the next higher counting place) "10 is the radix of the decimal system"
- S: (n) base, home (the place where you are stationed and from which missions start and end)
- S: (n) al-Qaeda, Qaeda, al-Qa'ida, al-Qaida, al Qaeda, al Qaida, Base (a terrorist network intensely opposed to the United States that dispenses money and logistical support and training to a wide variety of radical Islamic terrorist groups; has cells in more than 50 countries)
- S: (n) root, root word, base, stem, theme, radical ((linguistics) the form of a word after all affixes are removed) "thematic vowels are part of the stem"
- S: (n) infrastructure, base (the stock of basic facilities and capital equipment needed for the

functioning of a country or area) "the industrial base of Japan"

- S: (n) base (the principal ingredient of a mixture) "glycerinated gelatin is used as a base for many ointments"; "he told the painter that he wanted a yellow base with just a hint of green"; "everything she cooked seemed to have rice as the base"
- S: (n) base (a flat bottom on which something is intended to sit) "a tub should sit on its own base"
- S: (n) base ((electronics) the part of a transistor that separates the emitter from the collector)

Verb

- S: (v) establish, base, ground, found (use as a basis for; found on) "base a claim on some observation"
- S: (v) base (situate as a center of operations) "we will base this project in the new lab"
- S: (v) free-base, base (use (purified cocaine) by burning it and inhaling the fumes)

Adjective

- S: (adj) basal, base (serving as or forming a base) "the painter applied a base coat followed by two finishing coats"
- S: (adj) base, baseborn, humble, lowly (of low birth or station (`base' is archaic in this sense)) "baseborn wretches with dirty faces"; "of humble (or lowly) birth"
- S: (adj) base ((used of metals) consisting of or alloyed with inferior metal) "base coins of aluminum"; "a base metal"
- S: (adj) base, immoral (not adhering to ethical or moral principles) "base and unpatriotic motives"; "a base, degrading way of life"; "cheating is dishonorable"; "they considered colonialism immoral"; "unethical practices in handling public funds"
- S: (adj) base, mean, meanspirited (having or showing an ignoble lack of honor or morality) "that liberal obedience without which your army would be a base rabble"- Edmund Burke; "taking a mean advantage"; "chok'd with ambition of the meaner sort"- Shakespeare; "something essentially vulgar and meanspirited in politics"
- S: (adj) base, baseborn (illegitimate)
- S: (adj) base (debased; not genuine) "an attempt to eliminate the base coinage"

Figure 3 WordNet Example

3.4 Identifying Good Indexing Terms

Information retrieval pioneer Hans Luhn [Luhn 1958] believed that the *resolving power* of terms in a collection of text would be greatest in the middle-frequency range (see Figure 4). In this context, *resolving power* is the ability of a term to differentiate between documents relevant and irrelevant to the query. Neither high frequency terms, which are spread through many if not all documents, nor low frequency terms whose usage is isolated to only a few documents, constitute good indexing terms.

Thus, a large part of the task ahead involves selection of terms, term surrogates, phrases, and synonyms that best reveal information content. But first we begin with elimination of words that are unlikely to be useful terms.

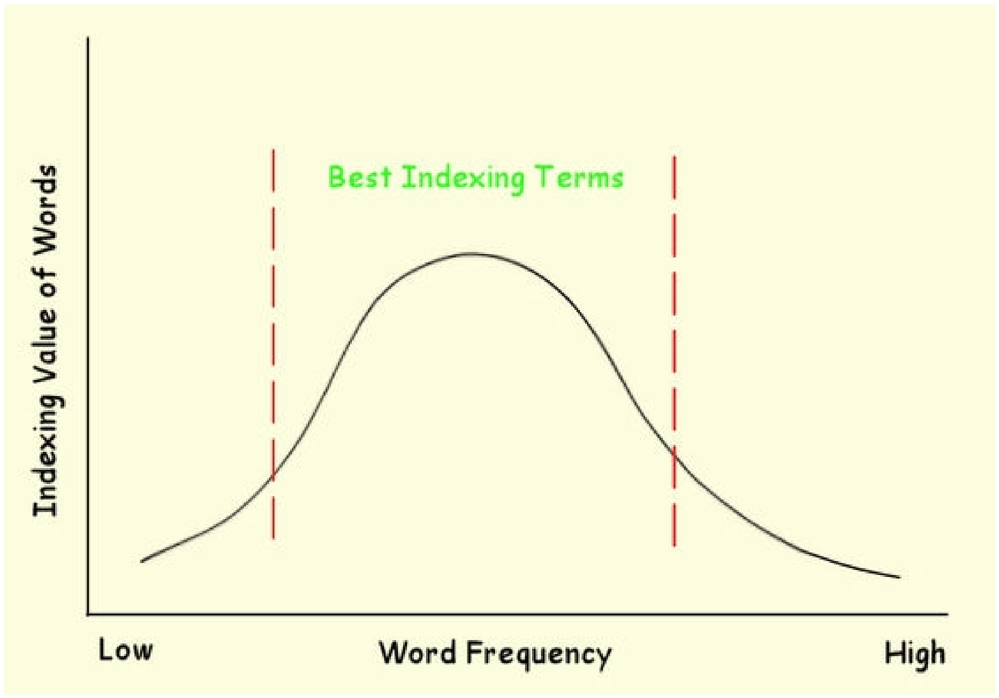


Figure 4 Best indexing terms

3.5 Zipf's Law

Zipf's Law states that the frequency ordered rank of a term in a document collection times its frequency of occurrence is approximately equal to a constant:

```
Frequency * rank ~= constant
```

where *Frequency* is the total number of times some term *W* occurs. Rank is the position number of the term when the terms have been sorted by *Frequency*. That is, the most frequently occurring term is rank 1, the second most frequently occurring term is rank 2 and so forth. Program 1 calculates Zipf's constants. The input to Program 1 was a dictionary created from the original collection. Words were unstemmed and stop words not removed.

```
#!/usr/bin/mumps
# zipf.mps Copyright 2014 Kevin C. O'Kane
# input is from sdin
# output is to sdout

    if '$data(%1) set d=1000
    else  set d=%1

    write $zd," Zipf Table Rank*Freq/d",!!
    for i=1:1 do
    . read a
    . if '$test break
    . set a=$zblanks(a)
    . set f=$piece(a," ",1)
    . set w=$piece(a," ",2)
    . set t=i*f/d
    . write $justify(t,6,0)," ",w,!
```

Program 1 Zipf's Law example

9 the	21 miss	22 or	22 my
11 and	22 is	22 freddy	23 been
12 of	22 all	23 by	22 mrs
15 to	21 be	22 have	22 came
17 a	20 mary	22 me	21 still
17 he	20 then	22 if	20 young
18 in	20 when	23 face	21 will
20 was	20 would	22 are	21 went
23 his	20 so	22 voice	21 time
24 her	21 julia	22 about	21 some
21 said	21 jane	22 no	21 do
21 she	21 out	22 eyes	21 while
23 had	22 browne	22 only	21 too
23 that	20 who	22 go	21 think
21 gabriel	21 which	21 this	21 stood
22 i	21 what	22 its	21 see
22 it	21 up	22 good	21 little
23 with	22 asked	22 back	22 how
24 for	20 them	22 an	22 himself
24 on	21 their	22 ivors	22 again
24 him	21 one	22 could	21 your
24 at	22 into	22 come	21 upon
23 aunt	21 there	22 over	21 two
24 as	22 malins	22 know	22 table
23 you	22 well	22 darcy	22 snow
21 mr	21 like	22 after	22 man
21 but	22 down	22 where	22 long
21 not	22 did	22 room	22 hand
21 kate	22 o	22 never	22 before
21 were	22 now	23 ladies	21 wife
21 from	21 we	23 gretta	22 why
21 they	22 very	22 old	22 three

Figure 5 Zipf constants - *The Dead*

Some Zipf results for James Joyce's short story *The Dead* are given in Figure 5 and the OHSUMED collection in Figure 6. A divisor is used to bring the constants, which have no units and correspond to no actual measurements, into a more readable range. The Zipf constant for these files differs due to their lengths.

369 of	403 during	393 new	447 treated
425 in	414 comment	398 protein	448 not
613 the	426 has	403 using	450 cases
733 and	423 comments	403 use	454 immunodeficiency
502 a	436 see	405 two	458 heart
555 with	428 or	411 syndrome	460 myocardial
561 to	437 effects	417 therapy	463 cardiac
514 the	428 in	422 report	465 acid
529 for	425 disease	426 virus	469 primary
426 patients	416 clinical	430 growth	473 following
429 on	399 blood	434 children	472 diagnosis
462 by	408 at	439 role	472 receptor
498 is	410 cells	433 determine	475 analysis
432 letter	408 effect	433 rat	476 management
417 was	417 acute	422 editorial	476 gene
407 from	423 to	420 pulmonary	479 carcinoma
425 were	408 associated	423 cancer	479 may
437 an	397 be	425 artery	478 we
414 human	400 case	426 an	482 care
365 study	400 studied	430 ventricular	484 patients
379 we	396 between	435 disease	481 liver
396 that	394 renal	437 response	484 [news]
393 after	389 chronic	439 activity	486 can
386 as	383 factor	439 normal	490 effect
394 are	380 patient	435 infection	493 bone
396 have	385 who	437 function	497 type
370 cell	391 this	442 which	497 its
381 treatment	383 coronary	441 studies	494 had
393 been	389 used	444 this	

Figure 6 Zipf constants - OHSUMED

From Zipf's Law we can hypothesize that if a term's usage in a document departs from it's expected usage, the term may have indexing value.

4 Database Searching

4.1 Linear Searching

At its most basic, a database may be searched by means of a linear or sequential search of a file of records. While this method is not practical for a real-time retrieval system, it has been used in the past prior to the wide availability of cheap direct access storage devices.

In early systems, before the availability of large scale direct access disks, when large scale data storage was mainly dependent upon magnetic tape, queries were accumulated during the day via primitive networks and batch processed against the database overnight. As each database record was scanned, each of the day's queries was processed against the record. Thus, the master database was scanned only once.

Results were transmitted back to originating sites at the end of the nightly processing. The Chemical Abstracts Service (CAS) used this method for a number of years. Users would interact with specially trained experts to formulate queries in the format and nomenclature of the CAS system and these queries would be transmitted by teletype to a central site in Ohio. Results would be returned by teletype the next day.

While the turn-around time on an individual query was a long, the number of queries that the system could handle per day was quite large and the cost per query quite low.

4.2 Inverted File Searching

In practice, while overnight batch searching of databases may be efficient, real-time access is the norm. To implement real-time database access what is needed is a method to quickly translate search keywords into pointers to the documents in which they occur. This lead to the development of systems based in inverted file systems.

An inverted file for information retrieval is one in which there is a directly addressable table that contains entries for each word in the indexing vocabulary. For each table entry there is a set of pointers to the documents that contain the words.

In a retrieval system there is a one-to-many relationship between a keyword and the documents that contain the keyword. That is, a given vocabulary word such as *computer* might be used in several thousand documents.

Consequently, in an inverted file system the entry for each word ordinarily points to one or more records that contain the actual pointers to the documents in the database containing instances of the keyword as shown in Figure 7.

Implementation of the inverted indices, however, can be done in several ways. Two common implementations are hash based indexing and tree based indexing.

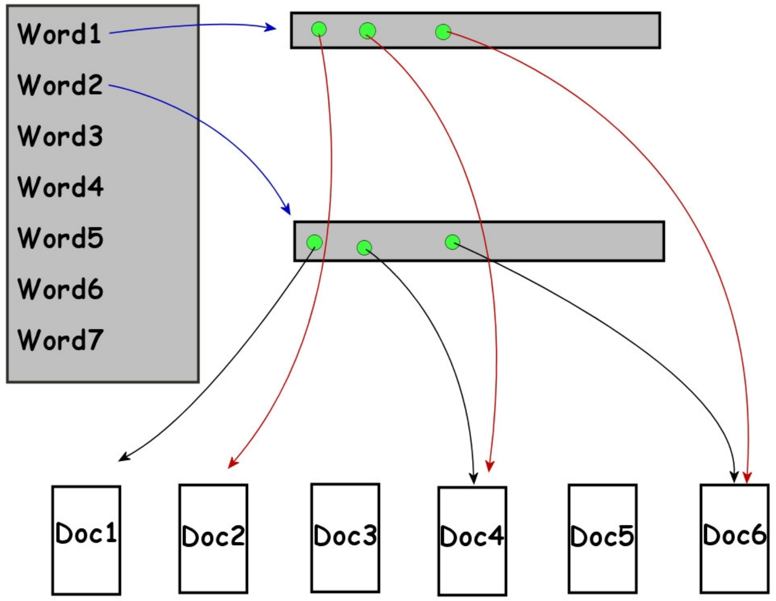


Figure 7 Inverted file organization

4.2.1 Hashed Indices

A hash based algorithm is direct key to address translation method that involves mathematically manipulating the search key in order to obtain an address in a table were the search key may reside. When two or more keys generate the same table address (a collision), there are several methods available to assign an alternative location for one of the colliding keys. The effectiveness of the technique is based on a number of factors however, in some cases, it can be very effective.

4.2.2 Tree Based Indices

In a tree based scheme, the set of keywords of the inverted index can be represented in a tree structure. Searching involves traversing the tree until the key is found or determined not to exist.

Some of tree based system have been used over the years include simple binary trees, balanced binary trees, weight balanced binary trees, multi-way trees, and several variations on B-trees. At present, B-tree based systems are the most widely used.

5 Vector Space Model

5.1 Overview

One popular approach to automatic document indexing, the vector space model, views computer generated document vectors as describing a hyperspace in which the number of dimensions (axes) is equal to the number of indexing terms. This approach was originally proposed by G. Salton [Salton 1968, 1971, 1983, 1992].

Each document vector is a point in that space defined by the distance along the axis associated with each document term proportional to the term's importance or significance in the document being represented. Queries are also portrayed as vectors that define points in the document hyperspace. Documents whose points in the hyperspace lie within an adjustable envelope of distance from the query vector point are retrieved. The information storage and retrieval process involves converting user typed queries to query vectors and correlating these with document vectors in order to select and rank documents for presentation to the user.

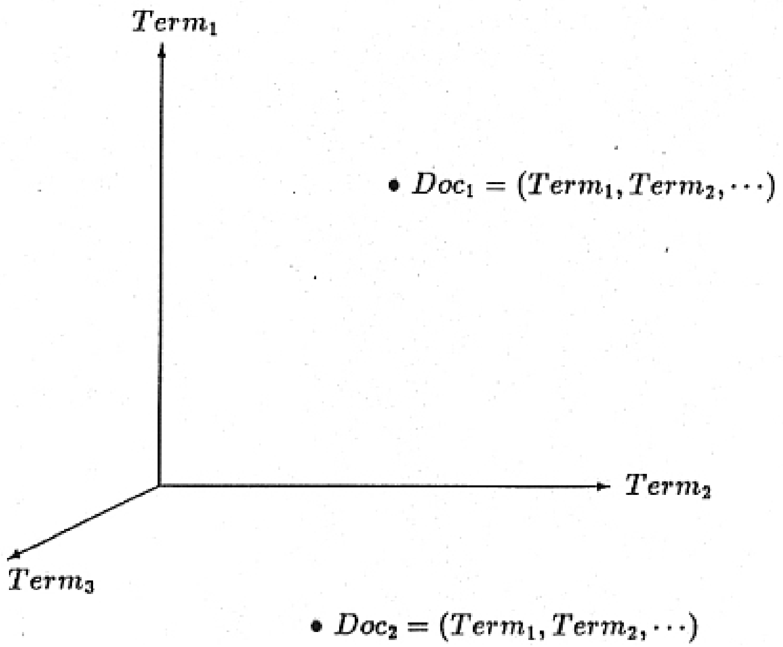

$Term_1$
$\bullet\ Doc_1 = (Term_1, Term_2, \cdots)$
$Term_2$
$Term_3$
$\bullet\ Doc_2 = (Term_1, Term_2, \cdots)$

Figure 8 Vector space model

Documents are viewed as points in a hyperspace whose axes are the terms used in the document vectors. The location of a document in the space is determined by the degree to which the terms are present in a document. Some terms occur several times while other occur not at all. Terms also have weights associated with their content indicating strength and this is factored into the equation as well.

Queries are converted to Query vectors and also treated as points in the hyperspace and the documents that lie within a set distance of the query are determined to satisfy the query.

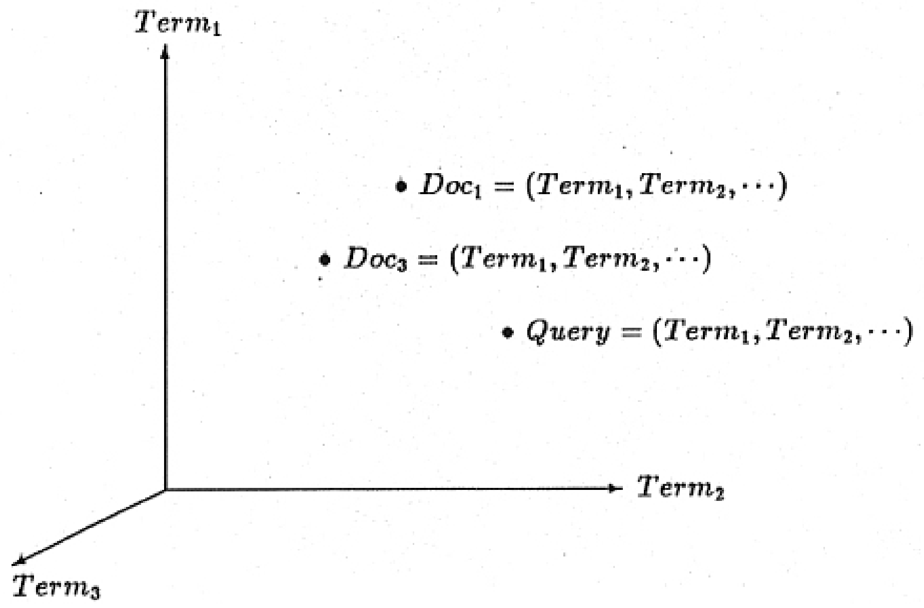

$Term_1$
$Doc_1 = (Term_1, Term_2, \cdots)$
$Doc_3 = (Term_1, Term_2, \cdots)$
$Query = (Term_1, Term_2, \cdots)$
$Term_2$
$Term_3$

Figure 9 Vector space queries

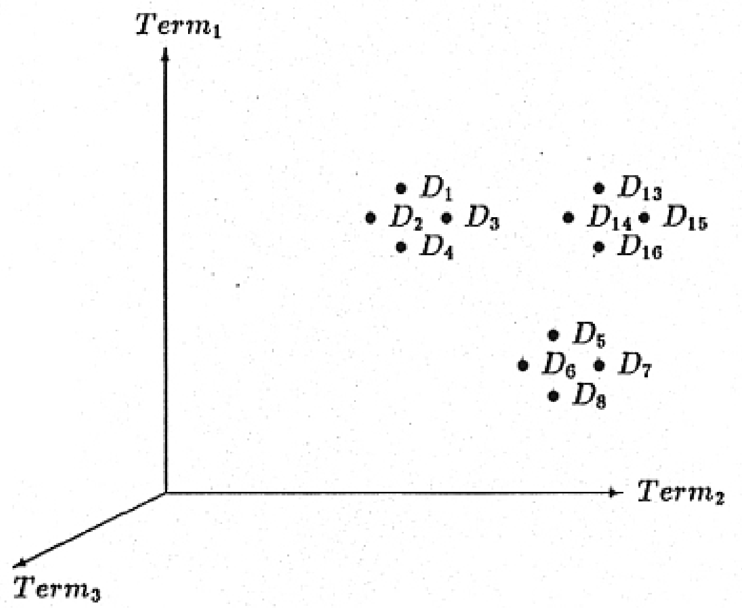


Figure 10 Vector space clustering

Another technique involved recognizing when documents are similar to one another through clustering. Clustering involves identifying groupings of documents and constructing a cluster centroid vector to speed information storage and retrieval. Hierarchies of clusters can also be constructed.

5.2 Basic Vector Similarity Functions

There are several popular formulae to calculate the distance between points in the hyperspace. One of the better known is the Cosine function [Salton 1983]. In this formula, the Cosine between points is used to measure the distance. Some of the common formulae are shown in Figures 11, 12, 13, 14, and 15 [Salton, 1983].

$$\mathrm{Sim}_1(Doc_i, Doc_j) = \frac{2\left[\sum_{k=1}^{t} (Term_{ik} \cdot Term_{jk})\right]}{\sum_{k=1}^{t} Term_{ik} + \sum_{k=1}^{t} Term_{jk}}$$

Figure 11 Dice Coefficient

$$\mathrm{Sim}_2(Doc_i, Doc_j) = \frac{\sum_{k=1}^{t} Term_{ik} \cdot Term_{jk}}{\sum_{k=1}^{t} Term_{ik} + \sum_{k=1}^{t} Term_{jk} - \sum_{k=1}^{t} (Term_{ik} \cdot Term_{jk})}$$

Figure 12 Jaccard Coefficient[1]

$$\mathrm{Sim}_3(Doc_i, Doc_j) = \frac{\sum_{k=1}^{t} (Term_{ik} \cdot Term_{jk})}{\sqrt{\sum_{k=1}^{t} (Term_{ik})^2 \cdot \sum_{k=1}^{t} (Term_{jk})^2}}$$

Figure 13 Cosine Coefficient

1 Jaccard, P. (1912). The Distribution of the Flora of the Alpine Zone. New Phytologist, 11, 37-50.

$$\mathrm{Sim}_4(Doc_i, Doc_j) = \frac{\sum_{k=1}^{t} (Term_{ik} \cdot Term_{jk})}{min(\sum_{k=1}^{t} Term_{ik}, \sum_{k=1}^{t} Term_{jk})}$$

Figure 14 Overlap Coefficient

$$\mathrm{Sim}_5(Doc_i, Doc_j) = \frac{\sum_{k=1}^{t} min(Term_{ik}, Term_{jk})}{\sum_{k=1}^{t} Term_{ik}}$$

Figure 15 Asymmetric Coefficient

The formulae in Figures 11, 12, 13, 14, and 15 calculate the similarity between **Doc$_\mathbf{i}$** and **Doc$_\mathbf{j}$** by examining the relationships between **term$_\mathbf{i,k}$** and **term$_\mathbf{j,k}$** where **term$_\mathbf{i,k}$** is the weight of term **k** in document **i** and **term$_\mathbf{j,k}$** is the weight of term **k** in document **j**. The examples in Figure 16 illustrate the application of the above[2].

Doc_i = (3,2,1,0,0,0,1,1)

Doc_j = (1,1,1,0,0,1,0,0)

$Sim_1(Doc_i,Doc_j)$= (2*6)/(8+4) -> 1
$Sim_2(Doc_i,Doc_j)$= (6)/(8+4-6) -> 1
$Sim_3(Doc_i,Doc_j)$= (6)/SQRT(16*4) -> 0.75
$Sim_4(Doc_i,Doc_j)$= 6/4 -> 1.5
$Sim_5(Doc_i,Doc_j)$= 3/8 -> 0.375

Figure 16 Example similarity coefficient calculations

2 Salton 1983, pg 202-203.

5.3 Document-Term Matrix

The vector space model, as noted above, is based on document vectors which initially[3] record the frequency of occurrence and, perhaps, the location of indexing terms that occur in each document.

A collection of document vectors is called a *document-term matrix*. A visualization is given in Figure 17 where each row is a document vector and each column represents an indexing word in the collection vocabulary[4].

	word 1	word 2	word 3	word 4	…	word x-3	word x-2	word x-1	word x
Doc 1	1	2	0	0	...	4	2	0	0
Doc 2	0	0	3	4	...	2	0	0	0
Doc 3	1	1	3	0	...	0	0	2	0
...	...	...	...	...	...	...	...	...	...
Doc n-2	4	0	0	1	...	0	1	0	1
Doc n-1	2	3	0	1	...	1	2	0	0
Doc n	0	0	1	3	...	0	0	2	4

Figure 17 Document- Term Matrix

In a collection of thousands, perhaps millions of documents, with a vocabulary of possibly tens of thousands of words, a fully mapped document-term matrix is impractically large.

In the case of the relatively small OHSU test collection there are approximately 300,000 abstracts (documents) and an initial candidate vocabulary of nearly 64,000 words[5].

If each matrix element required 4 bytes, this would mean a document-term matrix of approximately 300,000 * 64,000 or 19,200,000,000 elements and a storage requirement of 76,800,000,000 bytes.

3 In latter stages of processing, the frequency of occurrence of a word in a document will be replaced by an metric giving the importance of the word in the document.

4 Many common words and words whose distribution among documents is flat as opposed to clustered are not part of the indexing vocabulary.

5 Many of these terms will be discarded as bad indexing terms. In the OHSU database, about 5,000 indexing words remained after scoring words for indexing effectiveness.

Furthermore, this is only one matrix (usually several others are used).

However, each vector (row) usually only has a tiny number of word elements that actually exist, ten to twenty, in many cases. The remaining vector word cells are empty. In other words, most indexing terms don’t appear in most documents.

Furthermore, after scoring words for indexing effectiveness, only those terms that appear to be good indexing terms are retained while terms deemed to be of low indexing quality are discarded.

Ultimately, it the matrices are very sparse. That is, most cells contain NULL values.

An important question for the vector space model is: how do you represent the document-term and related matrices and vectors in a computer system given that the fully mapped visualization shown above in Figure 17 is not practical?

6 Implementation of the Vector Space Model

6.1 Database Requirements

While *ad hoc* implementations are always a possibility, an implementation of the vector space model conform to generally acceptable database models so that extensions. enhancements and improvements may be easily implemented. *Ad hoc* implementations often are limited in their ability to handle new requirements and are seldom transportable to different computing platforms.

Also, the database model should reflect, in an intuitive way, the conceptual vector space model. If the data base resembles the model, development of software is significantly less difficult.

The programming requirements for a workable database structure need to include the following:

1. The rows must be accessible consecutively, from low to high and vice-versa while individual rows must be accessible directly in random order.
2. Cell content must be directly addressable by row number and column word in consecutive or random order.
3. For a given row, the file access method must be able to report is a cell exists.
4. Rows and cells must be capable of being deleted.
5. Overall storage requirements must be minimized.

From a database point of view, there are only a few standard choices available to implement a model based on large, sparse matrices that offer reasonable performance. A Mumps based hierarchical implementation strategy is presented here.

6.2 Hierarchical Database Software

While database systems explicitly based on a tree or hierarchical model are not common at present, some still exist. One of the earliest and still available is IBM's Information Management System (IMS) originally released in 1966 for use inn NASA's Apollo Program.

Also developed in the late 1960s is Mumps (Massachusetts General Hospital Utility Multi-Programming System). Mumps was initially developed for use in medical records for which it is still widely used.

While Mumps is most often conceived of as a tree structured database, it can also be viewed as a multi-dimensional database. As such, it is well suited to the multidimensional vector space model. The vector space implementation, described in this document, is written in an open source version of Mumps.

6.3 Mumps Multi-Dimensional Database

A quick tutorial and interpreter, both free, for the Mumps language are available here:

`https://www.cs.uni.edu/~okane/`

Mumps is a relatively simple language and most people familiar with programming languages should be able to read it with minimal effort. However, as it's syntax is somewhat rigid, writing programs requires some training.

In the following, we survey the history of the language and highlight its main database features. There will be additional details in subsequent chapters as well.

Mumps (**M**assachusetts General Hospital **U**tility **M**ulti-programming **S**ystem) is a general purpose programming language environment that provides ACID (Atomic, Consistent, Isolated, and Durable) database access by means of program level subscripted arrays and variables. The Mumps database allows schema-less, key-value access to disk resident data organized as trees that may also be viewed as large, sparse multi-dimensional arrays.

6.4 Mumps History

Beginning in 1966, Mumps (also referred to as ***M***), was developed by Neil Pappalardo and others in Octo Barnett's lab at the Massachusetts General Hospital (MGH) on a PDP-7, the same architecture on which Unix was being implemented at approximately the same time.

Initial experience with Mumps was very positive and it soon was ported to a number of other architectures including the PDP-11, the VAX, Data General, Prime, and, eventually, Intel x86 based systems, among others. It quickly became the basis for many early applications of computers to the health sciences.

When Mumps was originally designed, there were very few general database systems in existence. The origin of the term 'database' itself dates from this period. Such systems as existed, were mainly *ad hoc* application specific implementations that were neither portable nor extensible. The notion of a general purpose database design was just developing[6].

For its database design, Mumps used a hierarchical or tree structured model because this closely matched the tree structured format of many medical records. An example is shown in Figure 18.

6 At about the same time, IBM's IMS (Information Management System), was being developed in connection with the NASA Apollo program. It was first placed into service in 1968 running on IBM 360 mainframes. IMS, which is still in use today, is, like Mumps, hierarchical in structure. IMS is reputed to be IBM's highest revenue software product.

To represent database trees in the language, they decided to use array references where each successive array index was part of a path description from the root of the array to intermediate and terminal nodes. They called these disk resident structures *global arrays*.

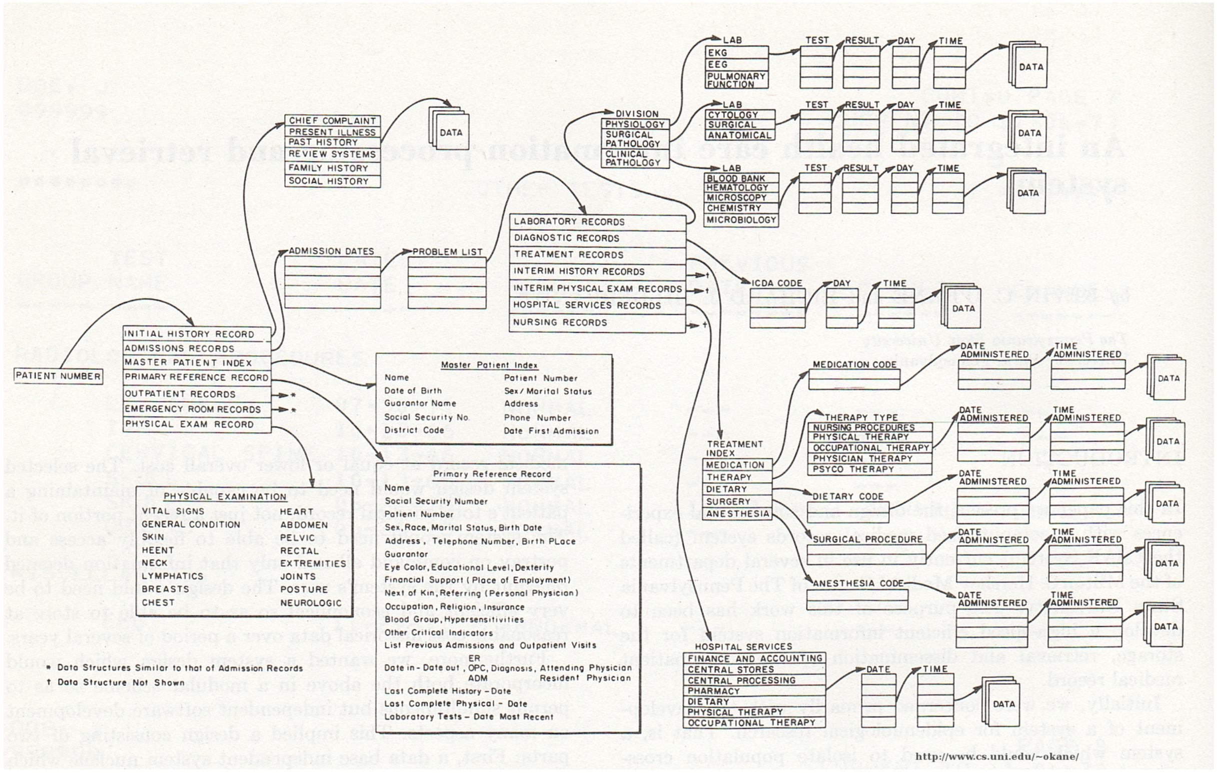


Figure 18 Patient Medical Record Tree

6.5 Mumps Global Array Database

Mumps implements, as an integral part of the language, a hierarchical and multi-dimensional database paradigm. When viewed as trees, data nodes can addressed as path descriptions in a manner which is easy for a novice programmer to master in a relatively short time. Alternatively, the trees can be viewed as sparse n-dimensional matrices of effectively unlimited size which are especially well suited to implement aspects of the Vector Space Model.

Mumps supports built-in string manipulation operators and functions that provide programmers with access to efficient methods to accomplish complex string manipulation and pattern matching operations.

The version of Mumps used here has been specially augmented with many additional builtin functions designed for document indexing applications.

In Mumps and similar hierarchically organized systems, the organization of the data can be viewed either as a tree with varying length paths from the root to an ultimate leaf node, or a multidimensional sparse matrix.

In Mumps, persistent data, that is, data that can be accessed after the program which created it terminates, is stored in *global arrays*. Global arrays are disk resident and are characterized by the following:

1. They are not declared or pre-dimensioned.
2. The indices of an array are specified as a comma separated list of numbers or strings.
3. Arrays are sparse. That is, if you create an element of an array, let us say element 10, it does not mean that Mumps has created any other elements. In other words, it does not imply that there exist elements 1 through 9. You must explicitly create these it you want them.
4. Array indices may be positive or negative numbers or character strings or a combination of both.
5. Arrays may have multiple dimensions limited by the maximum line length (nominally 512 characters but most implementations permit longer lengths).
6. Arrays may be viewed as either matrices or trees.
7. When viewed as trees, each successive index is part of the path description from the root to a node.
8. Data may be stored at any node along the path of a tree.
9. Not all nodes will have data. In fact, some trees may have no data stored.
10. Global array names are prefixed with the up-arrow character (^).

Syntactically, global arrays differ from local arrays only in that global array names are prefixed by an up-arrow character (**^**).

For example, consider an array reference of the form *^root("p2","m2","d2")*. This could be interpreted to represent a cell in a three dimensional matrix named *^root* indexed by the values **(***"p2","m2","d2")* or, alternatively, it could be interpreted as a path from the origin (*^root*) to a final (although not necessarily terminal) node *d2*.

In either the array or tree interpretation, values may be stored not only at an end node, but also at intermediate nodes. That is, in the example above, data values may be stored at nodes *^root, ^root("p2"), ^root("p2","m2")* as well as *^root("p2","m2","d2")* or, at none at all.

Because Mumps arrays can have many dimensions (limited by an implementation defined maximum line length), when viewed as trees, they can be of many levels of depth and these levels of depth may differ from one sub tree to another.

In Mumps, arrays can be accessed directly by means of a set of valid index vales or by navigation of a global array tree primarily by means of the built-in functions *$data()* and *$order()*. The first of these, *$data()*, reports if a node exists, if it has data and if it has descendants. The second, *$order()*, is used to navigate from one sibling node to the next (or prior) at a given level of a tree.

The diagram shown in Figure 19 was created by Program 2. In these, each successive index added to the global array description leads to a new node in the tree. Some branches go deeper than others. Some nodes may have data stored at them, some have no data. The *$data()* function which takes a global array reference as an argument, can be used to determine if a node has data and if it has descendants.

In the example in Figure 19, only numeric indices were used to conserve space. In fact, however, the indices of global arrays are often character strings.

In a global array tree, the order in which siblings appear in the tree is determined by the collating sequence, usually ASCII. That is, the index with the lowest overall collating sequence value is first branch at a given level of the tree, and the index with the highest value is last branch. The *$order()* function, which takes a global array reference as an argument, can be used to navigate from one sibling to the next at any level of the tree. The tree from Figure can be created with the code[7] shown Program 2.

In this example, note that several nodes exist but have no data stored. For example, the nodes *^root(1)*, *^root(8)* and *^root(32)* exist because they have descendants but they have no data stored at them. On the other hand, the node *^root(32,5)* exists, has data and has descendants. Node *^root(32,123)* has data but no descendants.

7 Mumps assignment statements require the keyword *set.*

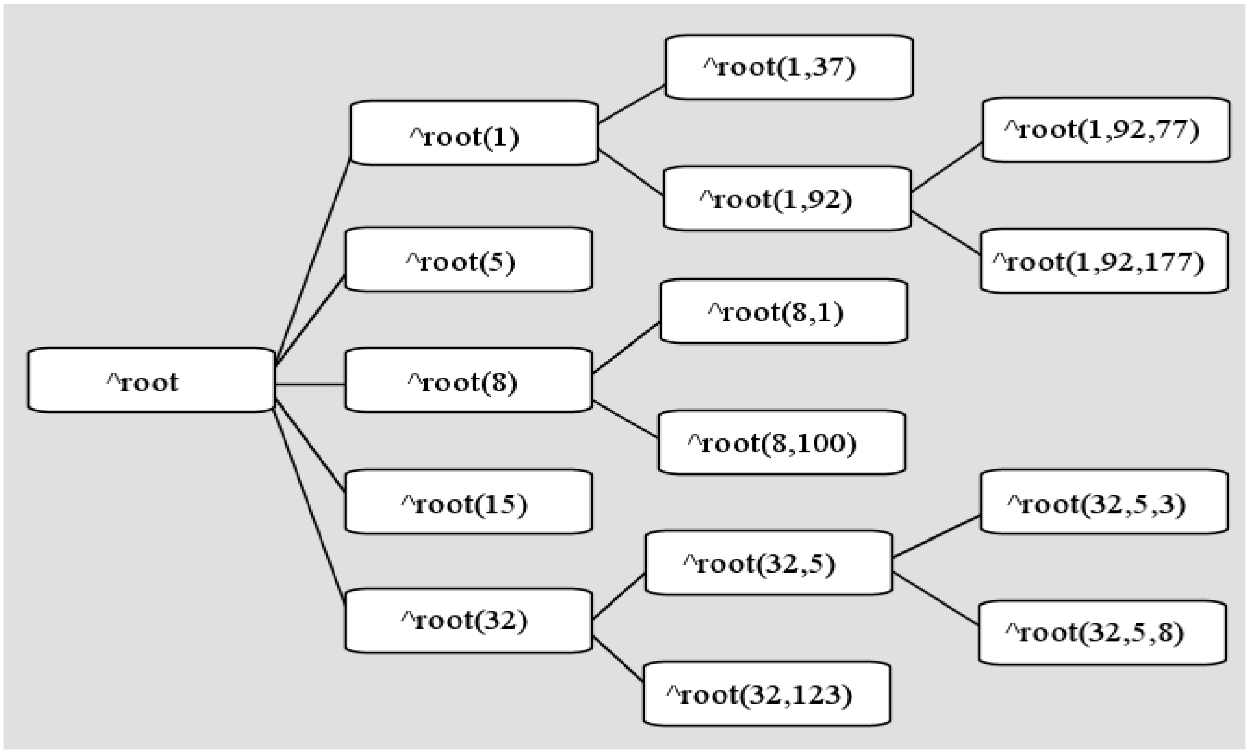


Figure 19 Global Array Tree

```
#!/usr/bin/mumps
  set ^root(1,37)=1
  set ^root(1,92,77)=2
  set ^root(1,92,177)=3
  set ^root(5)=4
  set ^root(8,1)=5
  set ^root(8,100)=6
  set ^root(15)=7
  set ^root(32,5)=8
  set ^root(32,5,3)=9
  set ^root(32,5,8)=10
  set ^root(32,123)=11
```

Program 2 Creating a Global Array

In some cases, an empty string is stored at a leaf node of a global array because the path description itself is the actual data. For example, consider the global array containing clinical laboratory tests seen in Program 3.

In this case, the first index is a patient id number, the second is the name of a lab test, the third is the date of the test and the fourth is the test result. In actuality, no further information is needed: the indices contain all the data.

A more realistic example can be seen in the Figure 20 which shows a section of the Medical Subject Headings (MeSH) as developed by the U.S. National Library of Medicine.

```
#!/usr/bin/mumps
  set ^lab(1234,”hct”,”05/10/2008”,38)=””
  set ^lab(1234,”hct”,”05/12/2008”,42)=””
  set ^lab(1234,”hct”,”05/15/2008”,35)=””
  set ^lab(1234,”hct”,”05/19/2008”,41)=””
```

Program 3 Global Array with Null Data

In Figure 20, a sub-tree of the National Library of Medicine MeSH hierarchy is represented along with the corresponding Mumps code to create the sub-tree.

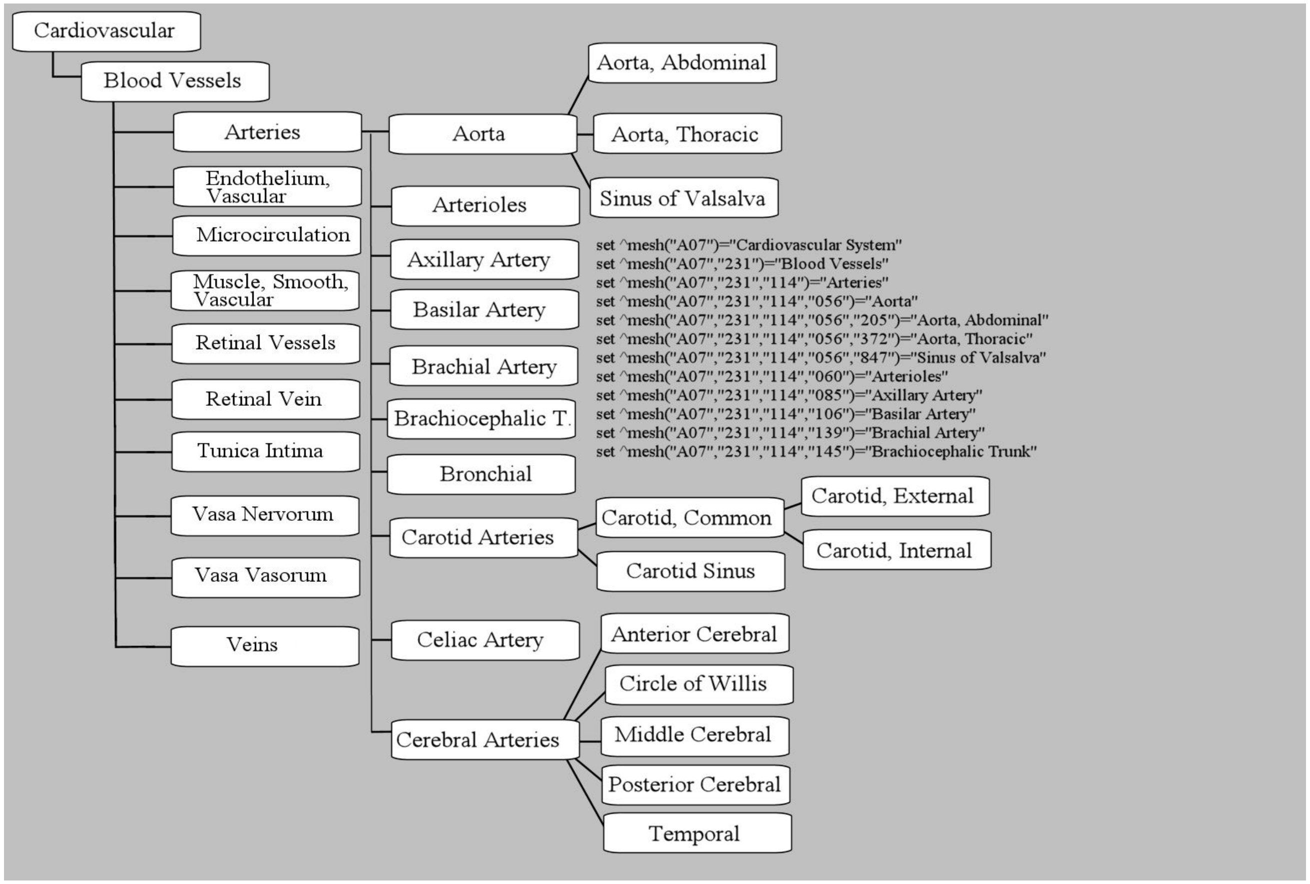
Quote marks around the numeric indices are not required except in cases where you want to preserve leading zeros as is the case in some nodes.

As noted above, data may be stored not only at fully subscripted terminal tree elements but also at other levels. For example, a three dimensional matrix named *mat1*, could be initialized as shown in Program 4. In this example, all the elements of a traditionally structured three dimensional matrix of 100 rows, 100 columns and

100 planes are initialized to zero. Note: the **for** command is the iterative loop command in Mumps. Its arguments are a loop variable, an initial value, an increment, and a final value.

```
#!/usr/bin/mumps
  for i=0:1:100 do
  . for j=0:1:100 do
  .. for k=0:1:100 do
  ... set ^mat1(i,j,k)=0
```

Program 4 Global Data Only at Leaf Nodes



```
set ^mesh("A07")="Cardiovascular System"
set ^mesh("A07","231")="Blood Vessels"
set ^mesh("A07","231","114")="Arteries"
set ^mesh("A07","231","114","056")="Aorta"
set ^mesh("A07","231","114","056","205")="Aorta, Abdominal"
set ^mesh("A07","231","114","056","372")="Aorta, Thoracic"
set ^mesh("A07","231","114","056","847")="Sinus of Valsalva"
set ^mesh("A07","231","114","060")="Arterioles"
set ^mesh("A07","231","114","085")="Axillary Artery"
set ^mesh("A07","231","114","106")="Basilar Artery"
set ^mesh("A07","231","114","139")="Brachial Artery"
set ^mesh("A07","231","114","145")="Brachiocephalic Trunk"
```

Figure 20 MeSH Tree as a Global Array

While global arrays are unique to Mumps, as a programmer, you will work with them as though they were ordinary arrays but the system interprets them as path descriptions in the system's external data files.

Global arrays may have both string and numeric indices as shown in Program 5.

```
#!/usr/bin/mumps
  set a=”1ST FLEET”
  set b=”BOSTON”
  set c=”FLAG”
  set ^ship(a,b,c)="CONSTITUTION"
  set ^captain(^ship(a,b,c))="JONES"
  set ^home(^captain(^ship(a,b,c)))="PORTSMOUTH"
  write ^ship(a,b,c) → CONSTITUTION
  write ^captain("CONSTITUTION") → JONES
  write ^home("JONES") → PORTSMOUTH
  write ^home(^captain("CONSTITUTION")) → PORTSMOUTH
  write ^home(^captain(^ship(a,b,c))) → PORTSMOUTH
```

Program 5 Global Array Indices

6.6 Software and Database Distribution

The experiments whose examples are shown here were written in both Mumps and the related Multi-Dimensional and Hierarchical (MDH) C++ class library.

The distribution is provided as a compressed tar file (*.tgz*) for Linux organized as follows:

1. The Mumps languages processors including:
    a. A legacy Mumps interpreter written in C
    b. A collection of C++ code that includes:
        i. An expanded Mumps interpreter
        ii. A Mumps compiler
        iii. A C++ Mumps functionality class library
2. Indexing and GUI software organized as:
    a. Mumps code that can be interpreted or compiled.
    b. C++ version of the code that uses the MDH class library.

c. A collection of GTK/Glade/Mumps code to create GUI retrieval apps.

6.7 The Experimental Document Database

The experimental text collection provided in the distribution and used in many of subsequent examples and experiments is the OHSU MEDLINE Data Base which was obtained from the TREC-9 conference. TREC (Text REtrieval Conferences) are annual events sponsored by the National Institute for Standards and Technology (NIST). These data sets are (as of March 2022) at:

https://trec.nist.gov/data.html

The original OHSUMED data set can be found here:

https://trec.nist.gov/data/t9_filtering.html

The TREC-9 Filtering Track data base consisted of a collection of medically related titles and abstracts which requires the following disclosures:

"... The OHSUMED test collection is a set of 348,566 references from MEDLINE, the on-line medical information database, consisting of titles and/or abstracts from 270 medical journals over a five-year period (1987-1991). The available fields are title, abstract, MeSH indexing terms, author, source, and publication type. The National Library of Medicine has agreed to make the MEDLINE references in the test database available for experimentation, restricted to the following conditions:

1. The data will not be used in any non-experimental clinical, library, or other setting.

2. Any human users of the data will explicitly be told that the data is incomplete and out-of-date.

The OHSUMED document collection was obtained by William Hersh (hersh@OHSU.EDU) and colleagues for the experiments described in the papers below:

Hersh WR, Buckley C, Leone TJ, Hickam DH, OHSUMED: An interactive retrieval evaluation and new large test collection for research, The Proceedings of the 17th Annual ACM SIGIR Conference, 1994, 192-201.

Hersh WR, Hickam DH, Use of a multi-application computer workstation in a clinical setting, Bulletin of the Medical Library Association, 1994, 82: 382-389. ..."

6.7.1 Modified OHSUMED File

For purposes of this text, the OHSUMED file was modified and edited into a revised format similar to that previously used by MEDLINE (the current MEDLINE format is similar but different).

The MEDLINE/PubMed data element (field) descriptions are at:

http://www.nlm.nih.gov/bsd/mms/medlineelements.html

```
 .I 54711
.U
88000001
.S
Alcohol Alcohol 8801; 22(2):103-12
.M
Acetaldehyde/*ME; Buffers; Catalysis; HEPES/PD; Nuclear Magnetic Resonance; Phosphates/*PD; Protein Binding;
Ribonuclease, Pancreatic
/AI/*ME; Support, U.S. Gov't, Non-P.H.S.; Support, U.S. Gov't, P.H.S..
.T
The binding of acetaldehyde to the active site of ribonuclease: alterations in catalytic activity and effects of
phosphate.
.P
JOURNAL ARTICLE.
.W
Ribonuclease A was reacted with [1-13C,1,2-14C]acetaldehyde and sodium cyanoborohydride in the presence or absence of
0.2 M phosphate . After several hours of incubation at 4 degrees C (pH 7.4) stable acetaldehyde-RNase adducts were
formed, and the extent of their fo rmation was similar regardless of the presence of phosphate. Although the total
amount of covalent binding was comparable in the abse nce or presence of phosphate, this active site ligand prevented
the inhibition of enzymatic activity seen in its absence. This protec tive action of phosphate diminished with
progressive ethylation of RNase, indicating that the reversible association of phosphate wit h the active site lysyl
residue was overcome by the irreversible process of reductive ethylation. Modified RNase was analysed using 1 3C
proton decoupled NMR spectroscopy. Peaks arising from the covalent binding of enriched acetaldehyde to free amino
groups in the ab sence of phosphate were as follows: NH2-terminal alpha amino group, 47.3 ppm; bulk ethylation at
epsilon amino groups of nonessential lysyl residues, 43.0 ppm; and the epsilon amino group of lysine-41 at the active
site, 47.4 ppm. In the spectrum of RNase ethylated in the presence of phosphate, the peak at 47.4 ppm was absent.
When RNase was selectively premethylated in the presence of phosphate, to block all but the active site lysyl
residues and then ethylated in its absence, the signal at 43.0 ppm was greatly diminished, an d that arising from the
active site lysyl residue at 47.4 ppm was enhanced. These results indicate that phosphate specifically protec ted the
active site lysine from reaction with acetaldehyde, and that modification of this lysine by acetaldehyde adduct
formation res ulted in inhibition of catalytic activity.
.A
Mauch TJ; Tuma DJ; Sorrell MF.
```

Figure 21 Original OHSUMED format

The meaning of the codes used in Figure 21 are shown in Figure 22.

A sample of the original format of the input file is given in Figure 21 and the modified file format is shown in Figure 23. Some fields from the original file were not used in the modified.

The modified file is used to display results during actual retrieval.

```
1 .I     sequential identifier
2 .U     MEDLINE identifier (UI)
3 .M     Human-assigned MeSH terms (MH)
4 .T     Title (TI)
5 .P     Publication type (PT)
6 .W     Abstract (AB)
7 .A     Author (AU)
8 .S     Source (SO)
```

Figure 22 OHSUMED Codes

```
STAT- MEDLINE
MH    Acetaldehyde/*ME
MH    Buffers
MH    Catalysis
MH    HEPES/PD
MH    Nuclear Magnetic Resonance
MH    Phosphates/*PD
MH    Protein Binding
MH    Ribonuclease, Pancreatic/AI/*ME
MH    Support, U.S. Gov't, Non-P.H.S.
MH    Support, U.S. Gov't, P.H.S.
TI    The binding of acetaldehyde to the active site of ribonuclease: alterations in
       catalytic activity and effects of phosphate.
AB    Ribonuclease A was reacted with [1-13C,1,2-14C]acetaldehyde
      and sodium cyanoborohydride in the presence or absence
      of 0.2 M phosphate. After several hours of incubation
      at 4 degrees C (pH 7.4) stable acetaldehyde-RNase adducts
      were formed, and the extent of their formation was
      similar regardless of the presence of phosphate. Although
      the total amount of covalent binding was comparable
      in the absence or presence of phosphate, this active
      site ligand prevented the inhibition of enzymatic activity
      seen in its absence. This protective action of phosphate
      diminished with progressive ethylation of RNase, indicating
      that the reversible association of phosphate with the
      active site lysyl residue was overcome by the irreversible
      process of reductive ethylation. Modified RNase was
      analysed using 13C proton decoupled NMR spectroscopy.
      Peaks arising from the covalent binding of enriched
      acetaldehyde to free amino groups in the absence of
      phosphate were as follows: NH2-terminal alpha amino
      group, 47.3 ppm; bulk ethylation at epsilon amino groups
      of nonessential lysyl residues, 43.0 ppm; and the epsilon
      amino group of lysine-41 at the active site, 47.4 ppm.
      In the spectrum of RNase ethylated in the presence
      of phosphate, the peak at 47.4 ppm was absent. When
      RNase was selectively premethylated in the presence
      of phosphate, to block all but the active site lysyl
      residues and then ethylated in its absence, the signal
      at 43.0 ppm was greatly diminished, and that arising
      from the active site lysyl residue at 47.4 ppm was
      enhanced. These results indicate that phosphate specifically
      protected the active site lysine from reaction with
      acetaldehyde, and that modification of this lysine
      by acetaldehyde adduct formation resulted in inhibition
      of catalytic activity.
```

Figure 23 OHSUMED MEDLINE Modified Format

The meaning of the codes used in Figure 23 are given in Figure 24.

```
1 MH - MeSH heading term
2 TI - title
3 AB - abstract.
4 All data fields begin in column 7 and all descriptors begin in column 1
5 Each entry begins with the text STAT- MEDLINE
```

Figure 24 MEDLINE Codes

This revised file is named *ohsu.medline* in the distribution.

6.7.2 Document Preprocessing

Whatever the format of the document source file, it must be converted to the format used by the indexing software.

The indexing software input file format is one large file where:

1. Each document consists of one (possibly very long) line.
2. The text has been converted to lower case ASCII.
3. The first token on each line is a disk offset into the original document text file of the start of the data associated with the document.
4. The second token is a document number in ascending order beginning with one.
5. Each token is separated from the next by a blank.

An example is given in Figure 25 for documents 21 and 22 whose original information in the source document file is located at offsets 21335 and 22577, respectively.

```
21335 21 comparison of the effects of atropine and glycopyrrolate on cognitive function following general
anaesthesia. tests of orientation, concentration and short-term visual memory were used to assess 72 patients 1 day
before, and 2 days after, elective major surgery. patients were premedicated with papaveretum and either atropine or
glycopyrrolate, before receiving a standard general anaesthetic. those who had received atropine showed significant
postoperative short-term memory deficit (p less than 0.01), but no change in orientation or concentration. those who
had been given glycopyrrolate showed no significant cognitive changes after surgery. as glycopyrrolate does not
cross the blood-brain barrier freely, these findings support the involvement of central cholinergic mechanisms in
the deterioration of cognitive function in the postoperative period.

22577 22 use of simple tests to determine the residual effects of the analgesic component of balanced anaesthesia.
in order to evaluate simple means of determining the rate of recovery after general anaesthesia, the usefulness of
the critical flicker fusion threshold test, the maddox wing apparatus and the visual analogue scale were compared.
the postanaesthetic recovery score was used as a reference. two patient groups (n = 15 in each) received, in a
randomized double-blind study, a similar balanced anaesthesia for caesarean section, except that the analgesic
component was either fentanyl 2.5 micrograms kg-1 i.v. or buprenorphine 7.5 micrograms kg-1 i.v. maddox wing
apparatus and visual analogue scale were sensitive enough to differentiate between the postanaesthetic residual
effects of the two opioids, but critical flicker fusion threshold and, especially, postanaesthetic recovery score
were insensitive in this respect. there was no difference between the two patient groups in mean arterial pressure
and heart rate. our results show that the residual effects of different kinds of opioids as an analgesic component
of balanced anaesthesia can be differentiated using simple means like maddox wing apparatus and visual analogue
scales.
```

Figure 25 Final Input File Format

The *bash* script file *index.script* (see Chapter Error: Reference source not found) invokes a program (*reformat.mps* as seen in Program 6*)* that converts the file in the format shown in Figure 23 to that shown in Figure 25.

The file created by *reformat.mps* is *ohsu.converted* a sample of which is in Figure 25. This is the input to the indexing software.

Any document collection converted to the format shown in Figure 25 can be indexed by the indexing system.

```
#!/usr/bin/mumps
# Copyright 2014, 2022 Kevin C. O'Kane

# reformat.mps March 29, 2022

       zmain

       set M=$zgetenv("max_docs")
       if M="" set M=1000000
       open 1:"titles.list,new"
       kill ^map

       set D=0

       for  do  if D>M quit
       . set o=$ztell read line if '$test break
       . if $extract(line,1,2)="TI" do  quit
       .. set D=D+1
       .. use 1 write D," ",$extract(line,7,200),! use 5
       .. write off," ",D
       .. set ^map(D)=off
       .. set line=$zlower($e(line,7,2048))
       .. set line=$zblanks(line)
       .. for i=1:1 do
       ... s w=$p(line," ",i) if $l(w)=0 break
       ... w " ",w
       . if $extract(line,1,2)="MH" quit
       . if $extract(line,1,13)="STAT- MEDLINE" set off=o write:D'=0 ! quit
       . if $extract(line,1,2)'="AB" quit
       . for  do
       .. set line=$zlower($e(line,7,2048))
       .. set line=$zblanks(line)
       .. for i=1:1 do
       ... set w=$p(line," ",i) if $l(w)=0 break
       ... w " ",w
       .. read line if '$test break
       .. if line="" break
       .. if $find(line,"ABSTRACT") set line=$piece(line,"ABSTRACT",1)

       close 1
```

Program 6 Database Reformatting Program

7 Vector Space Document Storage and Indexing

7.1 Indexing Data Structures

At its simplest, document indexing involves scanning documents and identifying those terms or combinations of terms (phrases) that best represent the meaning and content of the documents and indexing the documents with those terms.

In practice, however, the process is more complicated as shown in Figure 26.

The main points in Figure 26 are:

1. The original corpus of documents is pre-processed into a format more easily manipulated by computer programs but with links to the original documents. Titles are usually stored in a separate file as these are normally the part of a document displayed during retrieval.
2. Documents are processed to remove common language words that convey no meaning from an indexing point of view.
3. Words in documents are reduced to semantic stems. In many cases, this involves prefix and suffix removal but in others it may involve a more complex semantic analysis. Words that are excessively short, long, or begin with numerics or special characters may also be removed at this time.
4. A dictionary is built giving a list of words remaining and their frequency of occurrence.
5. Based on word usage frequencies, documents are processed to remove very high and very low frequency terms as these are unlikely to be good indexing terms.
6. The dictionary is rebuilt to reflect the current vocabulary. Additionally, a document frequency dictionary is built giving the number of documents each word occurs in.
7. A *document-term matrix* is built where the row numbers are document numbers (arbitrarily assigned) and the columns are words. The value in a cell of the matrix gives the number of times a word appears in a given document. As each document contains only a small subset of the entire vocabulary, this is a very sparse (most cells have a value of zero) but potentially quite large matrix. For example, the OHSU medical collection used in the examples has nearly 300,000 documents and a vocabulary, excluding stop words, of nearly 50,000 words. This implies a matrix of 15,000,000,000 elements! In some systems, the document-term matrix also records the position in each document of each term occurrence.
8. A *term-document matrix* is calculated (the transpose of the document-term matrix) to provide inverted[8] file access to the documents.

8 That is, access to the documents by word rather than by document number.

9. Various term weights are calculated including the *inverse document frequency* weights and the *discrimination coefficients*.
10. The document-term and term-document matrices are recalculated to reflect the term weights and the frequencies of occurrence of words in individual documents.
11. A *term-term* matrix and a *term-term proximity matrix* are calculated identifying terms whose usage and proximity are similar for possible synonym identification and query supplementation.
12. A *document-document* matrix is built identifying documents whose word usage is similar.
13. A matrix of document clusters is built showing documents that are collectively related to one another. The centroid or average document vector from each cluster can be used as a surrogate for all documents in the cluster to improve retrieval speed.
14. Clusters of document clusters identified to provide a hierarchical view of the collection.
15. A matrix of term clusters is developed showing words whose usage is similar for possible use in augmenting queries and development of concept categories.

Documents are re-evaluated and original terms are replaced by concept tokens and the process re-iterates.

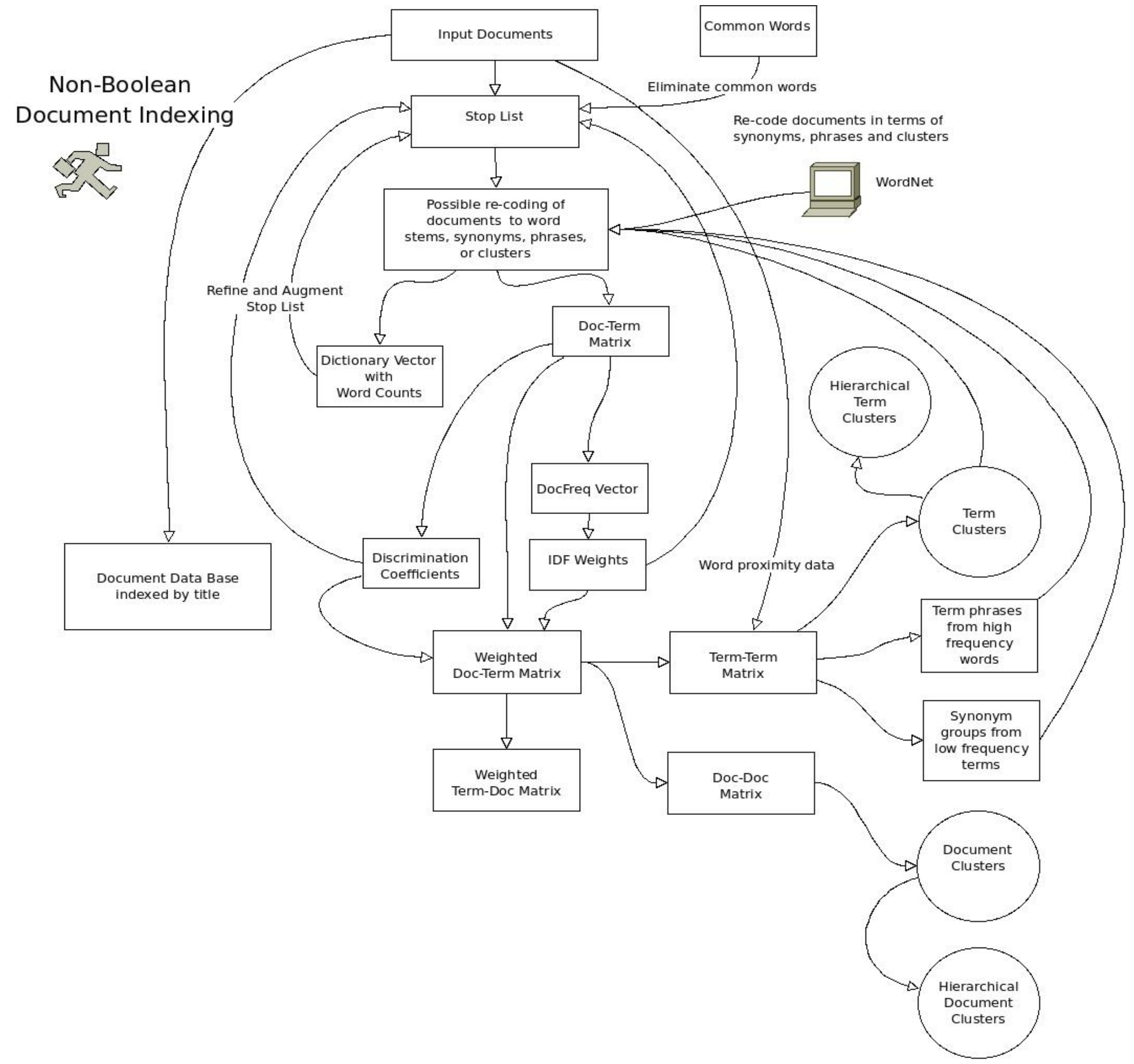

Non-Boolean
Document Indexing
Input Documents
Common Words
Eliminate common words
Stop List
Re-code documents in terms of synonyms, phrases and clusters
WordNet
Possible re-coding of documents to word stems, synonyms, phrases, or clusters
Refine and Augment Stop List
Doc-Term Matrix
Dictionary Vector with Word Counts
Hierarchical Term Clusters
DocFreq Vector
Term Clusters
IDF Weights
Word proximity data
Document Data Base indexed by title
Discrimination Coefficients
Term phrases from high frequency words
Weighted Doc-Term Matrix
Term-Term Matrix
Synonym groups from low frequency terms
Weighted Term-Doc Matrix
Doc-Doc Matrix
Document Clusters
Hierarchical Document Clusters

Figure 26 Overview of basic document indexing

8 Vector Space Software System Components

Software for document indexing systems can be organized much like an assembly line. Each document (or query in the case of retrieval) may be processed by a series of steps to produce a final result. For each of these steps, there are several possible algorithms that can be used, as we will see.

However, not all algorithms work equally well on all types of document collections. Some algorithms perform well on collections with well defined vocabularies, for example medicine and science, while others work better on collections of ordinary prose. An algorithm that works well on one collection type may work poorly on another.

Another overall concern is computational complexity. In some cases there are algorithms that perform well but are some computationally intensive that they are impractical when applied to large collections and thus impractical.

Thus, there is no one, correct way to build a document indexing or retrieval system. Instead, effective systems are built by selecting those functional building blocks that work best for the document collection at hand.

The optimal way to select the best combination of algorithms is by iterative experimentation where the designer examines the effects of different combinations of procedures and tuning parameters on overall performance.

Thus, in order to explore the effects of different combinations of indexing and retrieval algorithms, these have been implemented as collection of interoperable, modular programs that accompany this book.

Overall, the software modules implement the Vector Space Model first proposed by Salton [Salton 1968] which is discussed in detail below.

8.1 Software Components

The accompanying package includes:

1. a collection of indexing and retrieval programs written in Mumps and Bash script;
2. a corpus of nearly 300,00 medical journal abstracts; and
3. a Mumps language and database interpreter.

These are available for download at:

https://www.cs.uni.edu/~okane/

The modules in the package perform:

1. word frequency analysis,
2. stop list generation,
3. word stemming,
4. term weighting,
5. synonym detection,
6. phrase identification,
7. term clustering,
8. document clustering,
9. document hyper-clustering, and
10. several retrieval methods.

The programs build the following data structures:

1. document-term matrix,
2. term-document matrix,
3. term-term matrix
4. document-document matrix,
5. dictionary vectors giving:
    (a) word frequency,
    (b) document frequency,
    (c) Zipf's Law coefficients,
    (d) inverse document frequency weights [Salton 1968] and
    (e) discrimination coefficients [Willet 1985].

There are also programs to calculate:

1. term phrases,
2. term cohesion,

3. proximity weighted term similarities,
4. term clusters.
5. document clusters and
6. clusters of document clusters.

The package includes routines to retrieve documents based on:

1. simple sequential searches,
2. inverted file searches and
3. weighted inverted file searches using document similarity metrics such as Cosine [Salton 1983].

There also indexing routines to organize the documents by:

1. controlled vocabularies such as MeSH[9],
2. KWIC/KWOC[10] indices,
3. n-grams [Manning 1999] and
4. Soundex codes [US National Archives, 2007].

Selection of which programs to run, in what order, and with what parameters is determined by a large Bash script file named *index.script*.

9 National Library of Medicine Medical Subject Headings

10 Key Word In Context, Key Word Out of Context

8.2 Global Array Model

As mentioned above, one of the main data structures for the Vector State model is the *document-term matrix*. From this structure, we derive many other structures such as the transposed *term-document matrix*, the *term-term* matrix, the *document-document* matrix, *term-term* matrix and so forth.

In table form, the document-term matrix is a set of rows, also referred to as known as document vectors, where each row or vector represents a document. The columns denote terms from the indexing vocabulary.

A value in a cell, initially[11], is the number of times the column's word appeared in the document denoted by the row.

One of the initial stages of document processing involves identifying indexing terms from each document. The output of this step for document number 1003 from the OHSU collection is shown in Figure 27.

```
   Doc 1003: Enzymatic modification of glycocalyx in the treatment of experimental endocarditis due to viridans
streptococci.

   abundant (1) active (1) acts (1) administer (1) alpha (1) animal (2) antibiotic (1) associate (1) attack (1)
cardiac (1) combination (1) conclude (1) days (2) dextranase (5) digestion (1) endocardit (2) enzymatic (2) eradicate
(1) experiment (3) facilitate (1) failure (1) glycocalyx (5) infect (4) internal (1) linkage (1) modification (1)
organism (1) partial (1) penicillin (7) polysaccharide (1) presence (1) retard (1) sing (1) situte (1) sterilization
(2) streptococci (2) surface (1) test (1) therapy (2) treat (5) treatment (2) valve (1) vegetation (5) viridan (2)
vivo (1)
```

Figure 27 Document Vector 1003

In Figure 27, the first line contains the document number followed by the title. This is followed by a list of the words from the document, after common words have been removed and the remaining words reduced to stems, where possible. Following each word is the number of times it appears in the document. The example in Figure 27 is taken from the output file *document-term-matrix.txt.*

For a document 1003, a portion of the matrix row would appear as shown in Figure 28.

In the case of document 1003, there were 39 distinct words or terms after removal of stop list words and reducing, where possible, the words to common stems.

11 The number in a cell will be converted to a weight in later stages. That is, the weight of this vocabulary term in this document based on the number of time the word occurs in this document and the overall importance of the word as an indexing term in the collection as a whole.

In document 1003. the first three words, alphabetically, were *abundant, active,* and *acts* while the last three, alphabetically, were *vegetation, viridan* and *vivo*. These occurred 1, 1, 1, 5, 2 and 1 times, respectively, in the document and this is shown in Figure 28.

Document Number	abundant	active	acts	...	vegetation	viridan	vivo
...	...	...	...	...	...	...	...
1002	...	...	...	...	...	...	...
1003	1	1	1		5	2	1
1004	...	...	...		...	...	...
...	...	...	...		...	...	...

Figure 28 Document Term Matrix

In a Mumps global array, the contents for Figure would be represented as shown in Figure 29.

```
^doc(1003,”abundant” = 1
^doc(1003,”active”) = 1
^doc(1003,”acts” = 1
^doc(1003,”administer” = 1
.
.
.
^doc(1003,”vegetation”) = 5
^doc(1003,”viridan”) = 2
^doc(1003,”vivo”) = 1
```

Figure 29 Document Vector as a Global Array

In Mumps, if an element of a matrix does not exist, it takes no space. So, while the number of words in a document-term matrix may encompass 10s of thousands of words, an document vector with approximately 20 to 40 words, only occupies a small amount of space. It does not have 10s of thousands of NULL cells.

But the *^doc* matrix could be more detailed. In addition to storing the occurrence count for a word in a document, it could also store the relative location of each word in a document.

For example, the word *vegetation* appears five times in document 1003 at locations 24, 54, 59, 66, and 76. This can be represented in the *^doc* global array by adding a third index for each occurrence giving the location of the occurrence relative to the start of the document[12] as shown in Figure 30.

```
^doc(1003,"vegetation",24)=""
^doc(1003,"vegetation",54)=""
^doc(1003,"vegetation",59)=""
^doc(1003,"vegetation",66)=""
^doc(1003,"vegetation",76)=""
```

Figure 30 Word Positions in Global Arrays

The third index gives the relative location of each instance of the word. The relative location is all that is needed so we store an empty string at these nodes.

Now the matrix can be viewed both two dimensional (document, term) or three dimensional (document, term, position), depending upon whether we wish to access location information or not.

12 Position information is used to identify phrases based of frequecy of co-occurrence and proximity.

9 Basic Indexing

9.1 Vector Space Global Arrays

The following is a list of the main global arrays used to implement the vector space model in the associated software package:

1. *^dict(word)*

   This is a vector indexed by a vocabulary word giving the total number of total times the word occurs in the collection.

2. *^df(word)*

   This is a vector indexed by a vocabulary word giving the total number of documents the word appears in.

   For example, in an example database the term *patient* might occur 100 times in total in 50 different documents. The value of ^dict("patient") would be 100 and the value of ^df("patient") would be 50.

3. *^idf(word)*

   This is a vector indexed by a vocabulary word giving the *inverse document frequency* weight of the word in the collections as a whole. The *inverse document frequency* (IDF) is a measure of the importance of a word in the collection as a whole.

   So, for example, in a database of documents about medicine, the term *patient* might occur very frequently and in many documents. Consequently, it is not a good indexing term and its IDF weight will be low. Its usage is frequent and widely dispersed.

   On the other hand, the term *computer* in the same database, might occur infrequently and its usage clustered in a small set of documents. This term would have a high IDF weight because its usage is infrequent and concentrated.

   Term weight such as these (and others as seen below) give the weight of a term in the collection as a whole. The weight of a term in a specific document is usually the product of its weight in the collection and the number of times it appears in an individual document. Thus, a term of relatively low weight collection-wide may have a high document weight if it occurs frequently in a particular document.

4. *^mca(word)*

This is a vector indexed by a vocabulary word giving the *discrimination coeffiecient* of the word in the collection as a whole. Again, this is a measure of the importance of a word as an indexing term in the collection as a whole. A discrimination coefficient is an expensive calculation which measures the extent to which a word makes documents look more like one another. Terms that differentiate documents, that is, make them look less like one another, are better indexing terms.

MCA (modified centroid analysis) is a technique that approximates the value of the discrimination coefficient but at a much lower cost in computer resources [Willett 1985].

5. *^doc(d,word,pos)*

   This is the matrix known as the *document-term matrix*. It is indexed by the document number, the the string of a word appearing in the document and the position of an instance of the word in the document.

   The matrix can be viewed both as three dimensional and two dimensional. As a two dimensional matrix, *^doc(d,word), where d* is the document number and word is a *word,* the value stored at the node is either (1) the number of times *word* occurs in the document or, (2) the weight of *word* in the document.

   When the third index, *pos*, is present it gives the position(s) of each instance of *word* relative to the start of the document. No value is stored at the third level. The third index itself is the relevant data

   Rows of the document-term matrix are often referred to as *document vectors*.

6. *^index(word,d)*

   This matrix is the transpose of *^doc(d,word*) portion of document-term matrix. The position indicator is not stored in this matrix. It is referred to as the *term-document* matrix. The values stored at the nodes are same as the corresponding values in the *^doc(d,word)* matrix.

   An individual row of the term-document vector can be referred to as a *term vector.*

7. *^dd(d1,d2)*

   The *document-document* matrix giving a value indicating the similarity between two documents *d1* and *d2*.

8. *^tt(w1,w2)*

   The *term-term* correlation matrix giving the similarity of usage between word *w1* and word *w2*.

9. *^title(d)*

   This is a vector giving the text title of document *d*.

10. *^ttp(w1,w2,1)*

    The proximity term-term correlation matrix where *^ttp(w1,w2)* is the sum of the absolute values of the term proximities between word *w1* and *w2* and *^ttp(w1,w2,1)* contains the signed sum of the proximities.

    The proximity score between two words is the difference between their relative document positions as stored in the third index of *^doc(d,w1,pos).* A high score means that the words often appear closely to one another. The signed sum of the values indicates the preferred order of occurrence (which of the words is normally first).

9.2 Output Files

Figure 31 is a list of the output files created by *index.script* and a brief description of each:

deleted-words-for-frequency.txt	A list of words that were deleted because their frequency of occurrence was too high or too low.
discrimination-coefficients.txt	A list of words and their discrimination coefficients sorted by discrimination coefficient.
document-clusters-by-title.txt	A list of documents titles and the document clusters to which they belong.
document-clusters.txt	A list of document clusters giving the cluster number, the most heavily weighted title in the cluster, the most prominent terms in the cluster along with their weights, and and the document numbers and titles of the documents in the clusters.
document-clusters-table.txt	a table of cluster numbers and the document numbers and titles of the documents in the cluster.
document-document-matrix.txt	A list of pairs of document numbers and the cosine of similarities between them if the similarities exceed a threshold. This is a printout of the *^dd(d1,d2)* global array.
document-term-matrix-weighted.txt	A list of documents and, for each, a list of the words occurring in the document and the number of occurrences of each word in each document.

hyperclusters.txt	Gives the calculated clusters of clusters table.
idf-deleted-documents.txt idf-deleted-words.txt	An alphabetic list of words and their IDF weights that were deleted as a result of falling below the minimum IDF cutoff. As a result of word deletion, some documents may have lost too many words and were also deleted. These are listed as well.
idf-weights.sorted.txt	A list of words and their Inverse Document Frequency weights sorted by IDF weight from high to low.
kwic.txt	Key word in context index for the collection.
kwoc.txt	Key word out of context index for the collection.
phrases.txt	Hyphenated phrases consisting ow two words that are closely related. These phrases are entered into the documents in which the original terms occurred.
soundex.txt	The soundex codes for the vocabulary. These are used to find similaryly sounding words
SQL.commands	SQL CREATE TABLE and INSERT commands to create a copy of the document-term matrix in an SQL relational database system.
term-clusters.txt	A list of word clusters consisting of a cluster number and the terms in the cluster.
term-document-matrix-weighted.txt	The transpose of the document-term-matrix-weighted.txt file.
term-frequencies.txt	A list of words along with the number of documents each occurs in and the total number of occurrences of each.
term-term-cohesion-similarities.txt	A list or terms and those other related terms whose cohesion score exceeds a threshold.
term-term-jaccard-similarities.sorted.txt	A list of terms and those other related terms whose jaccard similarity exceeds a threshold.
term-term-matrix.sorted.txt	A list of pairs of terms and whose similarity exceeds a threshold along with the similarity and the number of co-occurrences, and occurrences of each term.
term-term-proximity-similarities.txt	A list term pairs that co-occur, a proximity similarity score, and an indication of the order in which they typically appear..
term-term-term.txt	A list term pairs with second order term connections.
zipfs-law.txt	Zpipfs Law calculations for the vocabulary.

Figure 31 Output Files

9.3 Run Time Settings and Defaults

Figure 32 lists the environment variables internal to *index.script* that control thresholds used by various programs.

The Bash script file *indexin.script* is invoked as:

index.script document-count database-name

where *document-count* is the number of documents to process (default 1,000) and *database-name* is the name of the database to be used (default *ohsu*).

max_docs	Number of documents to process. Set by first command line argument or by default (1000).
DB	Database prefix set from the second command line argument or default of “ohsu”
Bash variables used to determine actions to be performed and their default values.A value of “yes” enables the action while a value of “no” disables the action.	
SQL="no"	Generate relational database SQL table creation commands.
SQLite=”no”	Create SQLite database for Mumps global arrays.
compile="yes"	Convert (yes) Mumps programs to C++ and compile and use the result or (no) execute the Mumps programs interpretively.
zipf="no"	Run the Zipfs Law calculation program.
kwic="no"	Run the KWIC (Key Word In Context) program.
discrim="no"	Run the discrimination coefficient program.
proximity="no"	Calculate the term-term proximity weights.
doc_doc_matrix="yes"	Calculate the document-document matrix.
document_cluster="yes"	Calculate document clusters and hyper clusters (requires document-document matrix).
tt_corelations="yes"	Calculate the term-term correlation materix.
Document-Term Matrix Settings	
min_word_length=4	Words whose length is less than this value will be discarded.
min_doc_vector_word_count=20	The minimum number of words in a document vector. Vectors with fewer words will be discarded.
min_collection_word_count=5	Minimum number of time a word must appear in the collection as a whole. Words whose frequency is fewer than this value will be discarded.

max_collection_word_count=$((max_docs/4))	Maximum number of times a word may appear in the collection as a whole. Words appearing more frequently will be discarded.
min_doc_word_count=5	Minimum number of documents a word must appear in. Words appearing in fewer documents will be discarded.
Inverse Document Frequency (IDF) settings	
min_idf=4	Delete words whose IDF value is less than this.
min_idf_vector_count=$min_doc_word_count	Delete document vectors of those documents which, after IDF pruning, have fewer than
Document Weighting	
min_doc_weight=10	Delete words with a document weight less than this value.
min_doc_word_count	After deleting words for weigh, delete document vectors with fewer than this number of words remaining. Value of this variable is set above.
Term-Term Matrix Settings	
min_tt_corelation=0.250	To be recorded as related, terms must have a this vales as their minimum correlation.
min_tt_cooccur=3	To be recorded as related, terms must co-occur at least this number of documents.
Term-Term Proximity Matrix Setting	
proximity_min_count=20	To be recorded as related, terms must have an accumulated proximity score of at least this value.
Document-Document Matrix Settings	
min_doc_doc_cooccur=5	To be recorded as related, documents must have at least this number of terms in common.
min_dd_cosine=0.25	To be recorded as related, two document

	vectors must have a cosine similarity score of at least this value.
Jaccard Term-Term Settings	
jaccard_min=0.001	Minimum Jaccard similarity for two terms to be recorded as similar.
Document Cluster Settings	
min_cluster_cosine=$min_dd_cosine	Minimum cosine similarity between a document and a cluster for inclusion of the document into the cluster.
Hyper-Cluster Settings	
hyper_min=.3	Minimum cosine similarity between cluusters for inclusion in an hyper-cluster.
hyper_wgt=.3	Minimum total weight for a cluster to be a candidate for inclusion into an hyper-cluster.
Term-Term Cluster Settings	
min_tt_cluster=.1	Minimum similarity between two terms for inclusion in a cluster.

Figure 32 Bash Run-time Variables

9.4 Format the Input Database

As noted above, the first step is to reformat the input database from the format shown in Figure 23 to that shown in Figure 25. This is done in Program 7.

Program 7:

1. Reads the input line.
2. Records the location in the original file of the location of each STAT- MEDLINE line that begins each document in a global array named ^*map(D)* indexed by the document number and discards the STAT-MEDLINE line.
3. Records the title of each document, preceded by the document number, in a file named *titles.lst.*
4. Reduces the text to lower case and removes extraneous blanks.

5. Ignores the lines with MH
6. Writes to the output *stdout* the document number, followed by a blank, followed by the file offset of the STAT- MEDLINE that began this document followed by the words of the title and abstract separated from one another by one blank.
7. When all text of the abstract has been written, a linefeed is written.

```
#!/usr/bin/mumps
# Copyright 2014, 2022 Kevin C. O'Kane

# reformat.mps March 29, 2022

       zmain

       set M=$zgetenv("max_docs")
       if M="" set M=1000000
       open 1:"titles.list,new"
       kill ^map

       set D=0

       for  do  if D>M quit
       . set o=$ztell read line if '$test break
       . if $extract(line,1,2)="TI" do  quit
       .. set D=D+1
       .. use 1 write D," ",$extract(line,7,200),! use 5
       .. write off," ",D
       .. set ^map(D)=off
       .. set line=$zlower($e(line,7,2048))
       .. set line=$zblanks(line)
       .. for i=1:1 do
       ... s w=$p(line," ",i) if $l(w)=0 break
       ... w " ",w
       . if $extract(line,1,2)="MH" quit
       . if $extract(line,1,13)="STAT- MEDLINE" set off=o write:D'=0 ! quit
       . if $extract(line,1,2)'="AB" quit
       . for  do
       .. set line=$zlower($e(line,7,2048))
       .. set line=$zblanks(line)
       .. for i=1:1 do
       ... set w=$p(line," ",i) if $l(w)=0 break
       ... w " ",w
       .. read line if '$test break
       .. if line="" break
       .. if $find(line,"ABSTRACT") set line=$piece(line,"ABSTRACT",1)

       close 1
```

Program 7 *reformat.mps*

9.5 Stop Lists

All indexing techniques need to determine which words or combination of words are the better indexing terms and which terms are poor indications of content. However, in all languages, some words can be eliminated from further consideration immediately based on their frequency of occurrence.

Such a list of words is called a *stop list*. A *stop list* is a list of words which are not used for indexing (sometimes referred to as a *null dictionary*).

For the most part, a stop list is composed of:

1. very high frequency terms conveying no real meaning for purposes document classification (words such as: *the, and, was,* etc.) or,
2. very low frequency words that are one-of-a-kind and unlikely to be important in real world applications.

The file *basic.stop.words* contains, one word per line, a list of about 700 such words.

Once a stop list has been constructed, there are two ways to use it in a program. One way is to read each stop word into a Mumps global array and then test each input text word to see if it is in the stop list global array.

In the Mumps interpreter there are two added functions, *$zStopInit()* and *$zStopLookup().* These use memory resident high speed C++ lookup containers. At the beginning of Program 8 (described in section 9.6), *$zStopInit()* creates the container and reads the stop list words from file *basic.stop.wirds* (one word per line) into it.

In the body of the program, *$zStopLookup()* returns true (1) if the argument string passed to it is in the container and false (0) otherwise.

9.6 Filtering the Database for Stems and Stop List Words

A step normally done in most systems, as noted above, involves reducing words to root stems. That is, finding a common semantic root term and replacing all the variants of it with that single root term. For example, *computer, computing, computers computation, computed, computational* and so forth all have *comput* as a common root.

Sometimes stemming can as simple as converting plural forms to singular, while in other cases it can involve prefix and suffix removal along with other modifications. Sometime spelling must be taken into account such a *center* and *centre, favor* and *favour* and so forth. Sometimes, several variant words without a common spelling root may be combined (such as aerial and antenna).

There are a number of methods to reduce words to their root stems and the process is sometimes error prone. This version of Mumps has a basic builtin function known as *$zstem().* It returns the root of a word by means of a table driven technique that seeks to remove common prefixes and suffixes. A more thorough method, however, would involve the more extensive semantic vocabulary studies such as those found in WordNet shown below (Chapter 3.3 on page 19).

The output of Program 7 is preprocessed by Program 8 which:

1. Reads the file of basic English stop list words.
2. Opens for output files:
    1. *stem-words.lst* which will contain all the instances of words once they have been stemmed one to a line.
    2. *wordlist.tmp* which will contain all the instances of unstemmed words, one to a line.
3. Reads the input from Program 7 from *stdin*.
4. Writes to *stdout* the document number and file offset contained in the first two words of the input file.
5. Removes from the input line instances of numbers and punctuation (*$translate()*).
6. Removes duplicate blanks from the line
7. For each word *w* from the modified line, beginning with word three:
    1. Removes any leading minus signs.
    2. Checks if the word length is less than the minimum. Discards if it is.
    3. Checks to see if the word is in the basic stop list and discards if it is.
    4. Determines the word has a stem (*$zstem()*).
    5. Creates an instance of the global array ^*stem(x,w)* (if one does not already exist) where *x* is the stem and *w* is the original word. This global array records, for each stem, the list of original words that resulted in the stem.
    6. Writes the stem to *stem-words.lst* and the original word to *wordlist.tmp* both one word per line. Then it writes the stem to *stdout* as part of a very long line similar to the input.
8. The program ends when there is no more input.

The end result is that the input database file of documents has been converted to stems with stop list words, punctuation and numerics removed.

The files *stem-words.tmp* and *wordlist.tmp* contain the instances of each word and stem for the entire database. That is, the number of times a word appears in *wordlist.tmp* is the total number of times it appeared in the collection as a whole and the number of time a stem appears in *stem-words.tmp* is the total number of times the stem occurred in the collection.

The *stdout* is written to *ohsu.stemmed.*

```
#!/usr/bin/mumps
# Copyright 2014, 2022 Kevin C. O'Kane

# preprocess.mps Mar 29, 2022

       zmain

       set M=$zgetenv("max_docs")
       if M="" set M=1000000

       set %=$zStopInit("basic.stop.words")

       open 1:"stem-words.tmp,new"
       open 2:"wordlist.tmp,new"

       set D=0
       set min=%1

       for  do
       . read line if '$test break
       . set D=$p(line," ",2) if D>M break
       . u 5 write $p(line," ",1,2)
       . set line=$translate(line,"0123456789""`~!@#$%^&*()_+=?>.<,:;'{[}]","")
       . set line=$zblanks(line)
       . for i=3:1 do
       .. s w=$p(line," ",i)
       .. if $l(w)=0 break
       .. if $l(w)<min quit
       .. if $e(w,1,1)="-" set w=$e(w,2,99)
       .. if $zStopLookup(w) quit
       .. set x=$zstem(w)
       .. set ^stem(x,w)=""
       .. write " ",x use 2 write w,! use 1 write x,! use 5
       . use 5 write !

       close 1
       close 2
```

Program 8 *preprocess.mps*

9.7 Enhanced Stop List

While some words are common to all stop lists (such as *are*, *is*, *the*, *etc*.), other words may be discipline specific.

For example, while the word *computer* may be a significant content word in a collection of articles about biology, it is a common term conveying little content in a collection dealing with computer science. Consequently, it is necessary to individually examine the vocabulary of each collection to identify additional discipline specific words to include in the stop list in addition to the basic set of words common to all disciplines.

In the version of the OHSUMED collection used, the total number of documents is 293,857 and the total vocabulary consists of about distinct 120,000 words after (1) stemming[13], (2) rejection of words less than three or longer than 25 characters, and (3) words beginning with numbers.

A small number of words have very high frequencies of occurrence compared to the remainder of the file while at the low end of the frequency spectrum, there were about 72,000 words that occur 5 or fewer times (60% of the total number of words).

Figure 33 gives a graph of overall word usage in the OHSUMED collection. The vertical axis gives the number of times the word occurs and the horizontal the word rank[14]. The most frequently occurring word occurred about 180,000 times. As rank increases, frequency of occurrence drops dramatically.

If we eliminate words with total frequency of occurrence of 5 or less and greater than 40,000 (the top ranking 101 words), this eliminates about half the words and results in a candidate vocabulary of about 64,000 words.

On the other hand, in a Wikipedia data base the vocabulary is very large. In the 179 MB sample used, there were 402,347 distinct words after stemming, rejection of word whose length was less than three or greater than 25, and rejection of words beginning with digits.

Figure 34 gives the frequency and rank of the 75 most frequently occurring Wikipedia words. As can be seen, a very small number of words have very high frequencies of occurrence.

13 Removal of common suffixes.

14 The most frequent word is rank 1, the next, rank 2 and so forth.

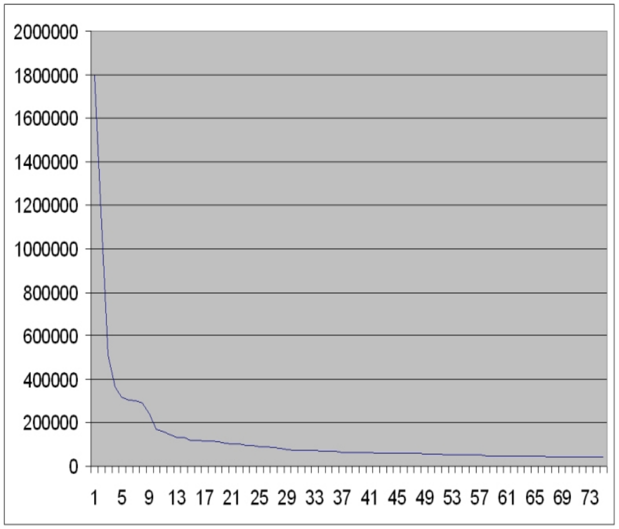
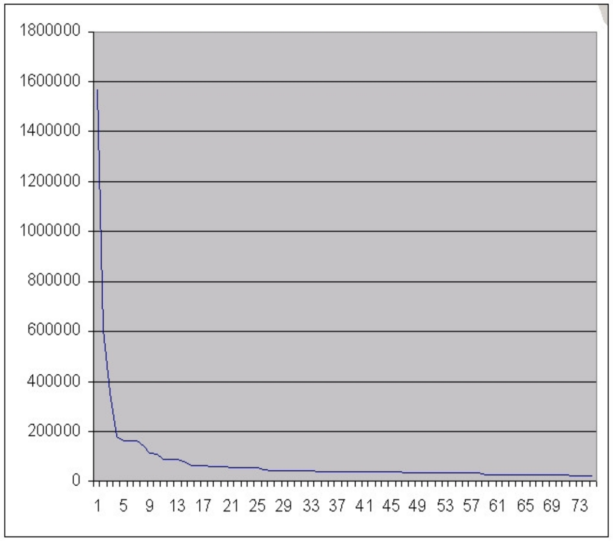
	

Figure 33 Frequency of top 75 OHSUMED words	Figure 34 Frequency of top 75 Wikipedia words

9.8 Building the Enhanced Stop List

The file *stem-words.tmp* from Program 8 is processed count the number of time each stem is used in the collection with the *bash* command line:

```
sort < stem-words.tmp | uniq -c > stemmed-vocabulary.tmp
```

which sorts the file *stem-words-tmp* alphabetically then counts the number of duplicate instances and write the result to *stemmed-vocabulary.tmp* and example from which is shown in Figure 35[15].

15 Note: the results in Figure 35 have been sorted and the sample shown displays a sample of low and high frequency stems. The actual file *stemmed-vocabulary.tmp* is not sorted.

```
     1 aa-induce
     1 aa/kg
     1 abai
     1 abat
     1 abate
     1 abbreviate
     1 abbreviation
     1 abdominal-thoracic
     1 abdominu
     1 aberran
     1 aberre
     1 abgs
     1 abim
     1 ablate
     1 ablative

   …

   776 change
   776 function
   780 mean
   824 clinical
   827 present
   833 increased
   838 letter
   874 blood
   909 norm
   957 level
   973 control
   975 rate
   990 year
  1104 disease
  1116 treatment
  1125 effect
  1131 case
  1181 result
  1379 cell
  1482 group
  1577 significant
  1944 study
  6096 patient
```

Figure 35 Example of *stemmed-vocabulary.tmp*

The file *stemmed-vocabulary.tmp* is passed as *stdin* input to the program *FrequencyFilter.mps* shown in Figure 36 which evaluates the frequencies of occurrence of stems based on command line provided high/low limits. Stems that are too frequent or not frequent enough are written to the file *stop.mps*. This file will be used as a second stop list file to remove additional words due to discipline specific frequency of occurrence.

A second file, *deleted-words-for-frequency.txt*, is written with the same information identifying which words were added to the new stop list.

```
#!/usr/bin/mumps
# Copyright 2014, 2022 Kevin C. O'Kane
# write out a new stop list consisting of high/low frequency words

# Feb 21, 2022

       zmain

       if '$d(%1) write "Missing parameter 1",! halt
       if '$d(%2) write "Missing parameter 2",! halt

       set min=%1
       set max=%2

       open 1:"stop.words,new"
       open 2:"deleted-words-for-frequency.txt,new"

       for  do
       . use 5 read a
       . if '$test break
       . set a=$zblanks(a)
       . set c=$p(a," ",1)
       . if c<min do
       .. use 1
       .. write $piece(a," ",2),!
       .. use 2 write "low ",$piece(a," ",2)," ",c,!
       .. quit
       . if c>max do
       .. use 1
       .. write $piece(a," ",2),!
       .. use 2
       .. write "high ",$piece(a," ",2)," ",c,!

       use 5
       close 1
       close 2
```

Figure 36 *FrequencyFilter.mps*

With regard to terms of very low frequency and terms of very high frequency, rather than elimination, it may be possible to *rescue* these terms based upon further analysis.

For example, low frequency terms that are determined to be semantically related to other low frequency terms may, collectively, if merged into a single semantic token and thus become a middle frequency term.

For example, highly specific medical terms relating to essentially the same condition when taken together may have a frequency of occurrence which places them in the middle frequency range.

Similarly, high frequency terms which by themselves convey little information may, when considered as phrases, be quite specific terms of middle frequency. For example: *programming* and *language* which separately are not very important in a database of computer abstracts. However, taken together as a phrase, they are somewhat more specific.

The new file *stop.words* is used by the program *DocumentTermMatrix.mps* to eliminate the newly identified stop list words.

9.9 Create Titles Vector

Depending on the value of *max_docs,* lines from *titles.list,* created earlier, are processed by the program *loadTitles.mps* as shown in Program 9 to produce a global array vector in the database named *^title(D)* which, for each document number *D* contains the text of the title for document *D*.

```
#!/usr/bin/mumps
# loadTitles.mps Feb 14, 2014
# Copyright 2014 Kevin C. O'Kane

      zmain

      for  do
      . read t
      . if '$test halt
      . set ^title($piece(t," ",1))=$piece(t," ",2,200)
```

Program 9 *loadTitles.mps*

9.10 Create the Document Term Matrix

The program *DocumentTermMatrix.mps* (see Program 10) reads as input (*stdin*) the file *ohsu.stemmed* created earlier and builds the global array vectors and matrices shown in Figure 37.

Words appearing in *stop.words* from above are ignored.

```
#!/usr/bin/mumps
# document-term-matrix.mps
# Copyright 2017, 2022 Kevin C. O'Kane

# Feb 21, 2022

      zmain

 kill ^df
 kill ^dict
 kill ^doc

 if '$data(%1) write "Missing parameter 1",! halt
 if '$data(%2) write "Missing parameter 2",! halt

 set wmin=%1
 set mindf=%2

 set %=$zStopInit("stop.words")

 set d=0

 open 1:"dict.tmp,new"
 open 2:"doc.tmp,new"
 open 3:"document-term-matrix.txt,new"

 for  do
 . use 5 read line
 . if '$t break
 . set off=$p(line," ",1),doc=$p(line," ",2),d=d+1,^doc(doc)=off
 . for j=3:1 do
 .. set w=$p(line," ",j)
 .. if $l(w)=0 break
 .. if $zStopLookup(w) quit
 .. set ^doc(doc,w,j)=""
 .. use 2 write doc," ",w,!
 .. use 1 write w,!

 use 5
 close 1
 close 2

 shell sort < dict.tmp | uniq -c > dict.sorted.tmp
```

```
open 1:"dict.sorted.tmp,old"
use 1

for  do
. read a
. if '$test break
. set a=$zblanks(a)
. set ^dict($piece(a," ",2))=$piece(a," ",1)

use 5
close 1

shell sort < doc.tmp | uniq -c > doc.sorted.tmp

open 1:"doc.sorted.tmp,old"

use 1

for  do
. read a
. if '$test break
. set a=$zblanks(a)
. set c=$p(a," ",1)
. set a2=$piece(a," ",2),a3=$piece(a," ",3)
. set ^doc(a2,a3)=c
. set ^index(a3,a2)=c

use 5
close 1

open 1:"df.tmp,new"
use 1

for d=$order(^doc(d)) do
. set c=0 for w=$order(^doc(d,w)) set c=c+1
. if c<wmin do
.. for w=$order(^doc(d,w)) kill ^index(w,d)
.. kill ^doc(d)
. else  do
.. for w=$order(^doc(d,w)) do
... write w,!

close 1
```

```
 shell sort < df.tmp | uniq -c | sort -nr > df.sorted.tmp

 open 1:"df.sorted.tmp,old"

 use 1
 for  do
 . read a
 . if '$test break
 . set a=$zblanks(a)
 . if +a<mindf break
 . set ^df($piece(a," ",2))=+a

 close 1
 use 5

 for w=$o(^df(w)) if '$data(^df(w)) kill ^dict(w)
 for w=$o(^dict(w)) if '$data(^df(w)) kill ^dict(w)

 for d=$order(^doc(d)) do
 . set c=c+1 for w=$order(^doc(d,w)) do
 .. if '$d(^dict(w)) kill ^doc(w) quit
 .. set c=c+1
 . if c<wmin do
 .. for w=$order(^doc(d,w)) kill ^index(w,d)
 .. kill ^doc(d)

 set c=0
 for d=$order(^doc(d)) set c=c+1
 set ^DocCount(1)=c

 open 1:"DocCount,new"
 use 1 write c,!

 close 1

# display doc-term matrix

 use 3

 for d=$order(^doc(d)) do
 . write "Doc ",d,": ",$e(^title(d),1,70),!
 . for w=$order(^doc(d,w)) do
 .. write w," (",^doc(d,w),") "
 .. if $x>60 write !
 . write !!

 close 3

# display document frequency vector

 open 1:"term-frequencies.unsorted.tmp,new"
 use 1

 for w=$order(^df(w)) write ^df(w)," ",^dict(w)," ",w,!

 close 1
 use 5

 shell sort -nr < term-frequencies.unsorted.tmp > term-frequencies.txt

 halt
```

Program 10 *DocumentTermMatrix.mps*

^df(stem)	For each stem, the number of documents the stem appears in.
^dict(stem)	For each stem, the total number of times it occurs in the collection.
^doc(D,stem)	For each document D, and each stem in D, the number of times the stem appears in document D.
^index(stem,D)	For each stem, and each document D the stem appears in, the number of times the stem occurs in document D.
^DocCount(1)	The total number of documents[16].

Figure 37 Global Arrays Created by *DocumentTermMatrix.mps*

16 Note: due to discarding words due to frequency and other factors, some documents may be discarded so the number of documents may decrease as processing continues.

document-term-matrix.txt	The document-term matrix. See Figure 39.
term-frequencies.txt	The document frequencies (*^df*) and total frequencies (*^dict*) of stems in the collection. See Figure 40.

Figure 38 *DocumentTermMatrix.mps* Output files

9.11 Normalized Word Frequencies

In the examples used here, the size of abstracts, with some exceptions, are approximately the same. That is, they have about the same number of words.

If, however, they differed significantly in length, it would be necessary to normalize the frequencies of word occurrence.

For example, if document 1 has 1,000 words and word A occurs 10 times and document 2 has 100 words and word A also occurs 10 times, the frequency of occurrence of word A needs to be adjusted so that it reflects its relative importance in both documents.

The simplest way to do this is to determine the average[17] number of words for documents in the collection and multiply, as appropriate, the frequencies of actual word occurrences accordingly.

This, from the example above, if the average document contains 500 words, the frequency for word A in document 1 should become 5 and the frequency for word a in document 2 should become 50.

17 Median is another possibility.

```
Doc 1018: Absorption of biliary calcium from the canine gallbladder: protection[18]
absorbe (1) absorption (4) addition (1) anion (1) bile (7) bili (4)
calcium (5) calcium-contain (1) canine (2) cation (1) charge (1)
comparison (1) component (1) concenter (3) concentration (2)
consistent (2) critical (1) declne (1) decrease (1) distribute (1)
distribution (1) effect (1) entry (1) epithelium (1) factor (2)
fast (1) fate (1) formation (1) free (2) gallbladd (6) gallstone (4)
great (1) greater (2) hour (1) imperme (1) importance (1) increased (2)
induce (1) ionize (1) likelihood (1) limit (1) major (1) minimum (1)
molecule (1) near (2) negative (1) neutralization (1) passive (2)
pathogenesis (1) pigment (3) precipit (2) prevention (1) previous (1)
proof (1) protect (1) ratio (1) regulate (1) report (1) result (1)
salt (3) specy (1) total (1)

Doc 1019: Partial aortic ligation: a hypoperfusion model of ischemic acute renal
acute (2) adenosine (2) aortic (1) artery (4) assay (1) assess (2)
azotemia (2) blood (6) cast (1) cessation (3) clearance (1) clinical (1)
close (1) compaare (1) compar (1) comparison (1) contrast (1)
creat (1) degree (1) depletion (2) depress (1) describe (1) develop (1)
difference (2) exist (1) experiment (2) extensive (1) failure (2)
flow (6) formation (1) glutathione (1) goal (1) hour (1) human (1)
hypoperfusion (5) indicate (1) induce (1) injury (2) interruption (1)
inulin (1) ischemic (6) just (1) later (1) left (1) ligation (2)
medull (1) mg/dl (2) minute (5) ml/min (1) model (7) necrosis (2)
occlusion (3) oxidant (1) pars (1) partial (1) perfusion (1)
pressure (1) proximal (1) rats (1) reli (2) renal (10) result (2)
reveale (1) severe (2) similarity (1) simulate (1) stress (1)
temporary (1) tissue (2) total (2) triphosphate (2) tubular (2)
typic (1) vascular (1) yielde (1)
```

Figure 39 Document Term Matrix Example

18 Note: titles truncated to 70 characters

```
72 165 occlusion
72 133 block
72 131 activation
72 127 line
72 124 practice
72 118 diameter
72 113 hemodynamic
72 112 recovery
71 98 interve
71 84 attempt
71 79 literature
71 76 compaare
71 138 metabolism
71 138 enzyme
71 135 program
71 124 position
71 123 malignant
71 121 recurrence
71 121 rabbit
71 117 systolic
71 108 research
71 103 return
70 86 dependent
70 78 adequate
70 73 play
70 166 spin
70 132 fami
70 129 perfusion
70 127 ischemic
70 108 selective
69 94 property
```

Figure 40 Document (^*doct*) and Total (^*df*) Frequencies Example

9.12 Assigning Word Weights

Words used for indexing vary in their ability to indicate content and, thus, their importance as indexing terms. Some words, such as *the*, *and*, *was* and so forth are worthless as content indications and we eliminated them from consideration immediately. Other words occur so infrequently that they are also unlikely to be useful as indexing terms. These were also eliminated. Other words, however, with a middle frequency of occurrence, are candidates to be indexing terms.

However, not all words are equally good indexing terms. For example, the word *computer* in a collection of computer science articles conveys very little information useful to indexing the documents since so many, if not all, the documents contain the word. The goal is to determine a metric indicating the ability of a word to convey information.

In the example in Figure 41, several weighting schemes are compared. The terms in Figure 41 are:

1. *^doc(i,w)* is the number of times term *w* occurs in document *i*;
2. *^dict(w)* is the number of times term *w* occurs in the collection as a whole;
3. *^df(w)* is the number of documents term w occurs in;
4. *NbrDocs* is the total number of documents in the collection; and
5. Function *$zlog()* is the natural logarithm;
6. The operation "\" is integer division.
7. Wgt1 is the number of times a word occurs in a document versus the number of time it occurs in a typical document. For example, if a word occurs twice as many times in a document than it typically does, Wgt1 will be 2.
8. Wgt2 uses what is known as the Inverse Document Frequency Weight (IDF) (discussed below). It is a measure of how widely distributed a word is in the collection. The IDF value of the word in the collection is multiplied by the number of times the word occurs in a document for the final weight of the term in a particular document.
9. Wgt3 combines Wgt1 and Wgt2. Words that occur more often in an individual document than they do in a typical document and are not widely distributed are given higher scores than words that are widely distributed and occur less frequently in a particular document than in the collection as a whole.
10. The MCA weight is the Modified Centroid Algorithm calculation method to calculate the Term Discrimination weight and will be discussed separately below.

In Figure 41 the document vectors for 20 documents (out of 1000) from computer science trade publications of the mid-80s are shown. Several term weighting schemes are shown.

While some knowledge of the state of the literature in the mid-80s would be useful, it should be clear that the word *computer* is not a good indexing term in a collection of articles about computers while a word such as *database* (databases were new then) is highly ranked.

```
Normalize [normal.mps] Sun Dec 15 13:08:59 2002


1000 documents; 29942 word instances, 563 distinct words

^doc(i,w)          Number times word w used in document i
^dict(w)           Number times word w used in total collection
^df(w)             Number of documents word w appears in
Wgt1               ^doc(i,w)/(^dict(w)/^df(w))
Wgt2               ^doc(i,w)*$zlog(NbrDocs/^df(w))+1
Wgt3               Wgt1*Wgt2+0.5\1

Word                   ^doc(i,w) ^dict(w) ^df(w)    Wgt1       Wgt2    Wgt3      MCA


[1]    Death of a cult. (Apple Computer needs to alter its strategy) (column)

apple                        4       261   112     1.716       9.757  17      -1.1625
computer                     4       706   358     2.028       5.109  10     -19.4405
mac                          2       146    71     0.973       6.290   6      -0.0256
macintosh                    4       210   107     2.038       9.940  20      -0.5855
strategy                     2        79    67     1.696       6.406  11      -0.0592

[2]    Next year in Xanadu. (Ted Nelson's hypertext implementations) Swaine, Michael.

document                     3       114    68     1.789       9.065  16       0.0054
operate                      3       269   184     2.052       6.078  12      -2.1852

[3]    WordPerfect. (WordPerfect for the Macintosh 2.0) (evaluation) Taub, Eric.

edit                         2       111    77     1.387       6.128   8      -0.0961
frame                        2         9     7     1.556      10.924  17       0.0131
import                       2        29    19     1.310       8.927  12       0.0998
macintosh                    3       210   107     1.529       7.705  12      -0.5855
macro                        3        38    24     1.895      12.189  23       0.1075
outstand                     1        10     9     0.900       5.711   5       0.0168
user                         4       861   435     2.021       4.330   9     -26.8094
wordperfect                  8        24     8     2.667      39.627  106       0.1747

[4]    Radius Pivot for Built-In Video an Radius Color Pivot. (Hardware Review) (new Mac monitors)(includes related
article on design of

built-in                     3        35    29     2.486      11.621  29       0.0678
color                        3        81    47     1.741      10.173  18       0.0809
mac                          2       146    71     0.973       6.290   6      -0.0256
monitor                      6        88    52     3.545      18.739  66       0.0946
resolution                   2        50    32     1.280       7.884  10       0.0288
screen                       2        92    62     1.348       6.561   9       0.0199
```

```
video                         4       106    61     2.302      12.188  28       0.0187

[5]    CrystalPrint Express. (Software Review) (high-speed desktop laser printer) (evaluation)

desk                          2       127    76     1.197       6.154   7      -0.1062
engine                        1        15    13     0.867       5.343   5       0.0282
font                          4       111    37     1.333      14.187  19       0.6350
laser                         3        61    27     1.328      11.836  16       0.2562
print                         3       140    66     1.414       9.154  13       0.0509

[6]    4D Write, 4D Calc, 4D XREF. (Software Review) (add-ins for Acius' Fourth Dimension database software)
(evaluation)

add-in                        2        97    38     0.784       7.540   6       0.5551
analysis                      2       179   139     1.553       4.947   8      -0.8492
database                      5       138    67     2.428      14.515  35       0.1832
midrange                      1         7     6     0.857       6.116   5       0.0218
spreadsheet                   2        75    44     1.173       7.247   9       0.1707
vary                          1         7     6     0.857       6.116   5       0.0107

[7]    ConvertIt! (Software Review) (utility for converting HyperCard stacks to IBM PC format) (evaluation)

converter                     2        24    13     1.083       9.686  10       0.0698
doe                           5        97    84     4.330      13.385  58      -0.1139
graphical                     2       307   171     1.114       4.532   5      -2.4079
hypercard                     4        25    13     2.080      18.371  38       0.1517
mac                           2       146    71     0.973       6.290   6      -0.0256
map                           2        17    10     1.176      10.210  12       0.1180
program                       4       670   334     1.994       5.386  11     -15.4832
script                        3        54    32     1.778      11.326  20       0.1239
software                      3       913   449     1.475       3.402   5     -30.7596
stack                         5        15     8     2.667      25.142  67       0.0700
```

Figure 41 Example word weights

9.13 Inverse Document Frequency Weight

One of the simplest word weighting schemes to implement is the Inverse Document Frequency weight. The IDF weight is a measure of how widely distributed a term is in a collection. Low IDF weights mean that the term is widely used while high weights indicate that the usage is more concentrated.

An IDF weight measures the weight of a term in the collection as a whole, rather than the weight of a term in a particular document.

In individual document vectors, the normalized frequency of occurrence of each term is multiplied by the IDF to give a weight for the term in the particular document. Thus, a term with a high frequency but a low IDF weight could still be a highly weighted term in a particular document, and, on the other hand, a term with a low frequency but a high IDF weight could also be an important term in a given document. The IDF weight for a term *W* in a collection of *N* documents is shown in Figure 42:

$$\log_2\left(\frac{N}{DocFreq_w}\right)$$

Figure 42 Inverse Document Frequency Weight

where $DocFreq_w$ is the number of documents in which term *W* occurs.

9.13.1 OHSU MEDLINE Data Base IDF Weights

The IDF weights for the OHSU MEDLINE collection were calculated based on the *^df()* vector created in Program Error: Reference source not found on page Error: Reference source not found.

Note that because of tuning parameters that set thresholds for the construction of the stop list and other factors, different runs on the document collection will produce variations in the values displayed. The IDF weights can range from lows such as shown in Figure 43

```
     0.189135 human
     0.288966 and
     0.300320 the
     0.542811 with
     0.737224 for
     0.793466 was
     0.867298 were
```

Figure 43 Low IDF Weights for OHSU MEDLINE

to highs such as shown in Figure 44

```
      12.590849 actinomycetoma
      12.590849 actinomycetomata
      12.590849 actinomycoma
      12.590849 actinomyosine
      12.590849 actinoplane
      12.590849 actinopterygii
      12.590849 actinoxanthin
      12.590849 actisomide
      12.590849 activ
      12.590849 activationin
```

Figure 44 High IDF Weights for OHSU MEDLINE

Note: for a given IDF value, the words are presented alphabetically. The OHSU MEDLINE collection has many terms that appear only once and, consequently, the IDF weights for the above, all with a document frequency of one, are the same.

9.13.2 IDF Weight Calculation

To calculate IDF weights we first build a document-term matrix *^doc(i,w),* as shown above, where *i* is the document number and *w* is a term. Each cell in the matrix contains the count of the number of times a term occurs in the document).

Next, from the document-term matrix, we construct a document frequency vector *^df(w)* where each element gives the number documents in which the term *w* occurs.

A basic program to calculate IDF weights is shown in Program 11. It assumes that the global arrays *^doc()* and *^df()* have been calculated (see Program Error: Reference source not found on page Error: Reference source not found).

```
#!/usr/bin/mumps
# idf.mps
# Copyright 2014, 2022 Kevin C. O'Kane
# Feb 24, 2022

 zmain

 set doc=^DocCount(1)

 if '$data(%1) write "missing idfmin - using 4.0" set min=4.0
 else  set min=%1

 for w=$order(^df(w)) do
 . set x=$zlog2(doc/^df(w))
 . if x<min quit
 . set ^idf(w)=$justify(x,1,3)

 halt
```

Program 11 IDF calculation

If a word has a very low or high IDF value, it may be a candidate for removal from the collection. Program 12 deletes words whose IDF value falls below a specified threshold. In these examples, we retain all high IDF words but this can be easily changed. Very high IDF values indicate a very low frequency of occurrence. Note that Program 12 also deletes documents if they have fewer than a predetermined (from a parameter) number of words in their vectors. This can also lead to words being deleted if, as a result, they are no longer contained in any document.

```
#!/usr/bin/mumps
# idf-cutoff.mps Feb 14, 2014
# Copyright 2014 Kevin C. O'Kane

# Feb 22, 2022

      zmain

      if '$data(%1) write "Missing parameter",! halt

      set wmin=%1
      set x=0

      open 3:"idf-deleted-words.txt,new"

      use 3

      for w=$order(^dict(w)) do
      . if $data(^idf(w)) quit
      . write w,!
      . kill ^idf(w) kill ^mca(w) kill ^dict(w)

      close 3
      use 5

      for d=$order(^doc(d)) do
      . for w=$order(^doc(d,w)) do
      .. if '$data(^idf(w)) kill ^doc(d,w),^index(w,d)

      open 3:"idf-deleted-documents.txt,new"

      use 3

      for d=$order(^doc(d)) do
      . set z=0
      . for y=$order(^doc(d,y)) set z=z+1
      . if z<wmin do
      .. for y=$o(^doc(d,y)) kill ^doc(d,y),^index(y,d)
      .. kill ^doc(d)
      .. write "Doc ",d," deleted count=",z,!
      .. set x=x+1

      close 3
      use 5

      for w=$order(^index(w)) if $order(^index(w,""))="" kill ^index(w)

      write !,?8,"IDF total docs deleted: ",x,!
```

```

     set D=0
     for d=$order(^doc(d)) set D=D+1
     set ^DocCount(1)=D
     write ?8,"IDF remaining documents: ",^DocCount(1),!

     open 1:"DocCount,new" use 1 write D,! close 1 use 5

     open 1:"idfWeights.tmp,new"

     use 1

     set wcount=0

     for w="":$order(^idf(w)):"" do
     . write ^idf(w)," ",w,!
     . set wcount=wcount+1

     use 5
     close 1

     write ?8,"IDF number of words remaining: ",wcount,!
     halt
```

Program 12 Deletion of words with low IDF scores

9.14 Discrimination Coefficients

Discrimination coefficients are another way to determine the importance of a term in the collection. Discrimination coefficients [Willet 1985, Crouch 1988, Salton 1983] measure the ability of terms to differentiate one document from another. They are calculated based on the effect a term has on overall hyperspace density with and without a given term.

If the space density is *greater* after a term is removed, that means the term was making documents look less like one another (a good discriminator) while a term whose removal *decreases* the density, thus making documents look more like one another, is a poor discriminator.

The discrimination values for a set of terms in a given document collection are comparable to the values for the IDF weights but not exactly.

The basic procedure to calculate discrimination coefficients involves calculating the average of pair-wise similarities between all documents in the space. This is the documents space density.

Then, for each word, the average pair-wise similarities of all the documents is re-calculated without the word. The difference in the averages is the term discrimination value for the term in question.

In practice, this is a very expensive weight to calculate unless speed-up techniques are used.

The centroid algorithm approaches [Crouch 1988], are an attempt to improve the speed of calculation. The exact calculation, where all pairwise similarity values are calculated each time, is of complexity on the order of *(N)(N-1)(w)(W)* where *N* is the number of documents, *W* is the number of words in the collection and *w* is the average number of terms per document vector.

Crouch [Crouch 1988] discusses several methods to improve the speed of this calculation. The first of these, the centroid approximate algorithm, involves calculating the similarities of the documents with a centroid vector representing the collection as a whole rather than all pair-wise similarities.

In the centroid approximation, a centroid vector for the entire collection is first calculated. A centroid vector is the average of all the document vectors and, by analogy, represents a point at the center of the hyperspace.

When we use a centroid vector, rather than calculate all the pair-wise similarities, we only calculate the similarities between each document and the centroid vector. The average of these similarities is taken to be the space density which, although not exactly the same as the full pairwise scheme, has been shown to be very similar [Crouch 1987].

The prime advantage of this method is thus, rather than calculating N*N document-document similarities, we calculate only N similarities. This improves the complexity to be on the order of *(N)(w)(W)* which represents a significant improvement in performance.

A further modification, called the Modified Centroid Algorithm (MCA), is possible. It is based on the centroid approximation from above but further reduces the overall complexity.

In the MCA, we calculate an initial space density in the same manner as the centroid approximation from above. However, in the MCA approach, we retain and store the individual contribution of each document to the initial total document space density. This requires additional space on the order of **N** to store the original **N** contributions.

Subsequently, when calculating the effect of a some term *i*'s removal on the document space density, the algorithm first subtracts from the total space density the original contribution of each document that contains term *i* (using the stored values) and then it adds to the space density the similarity between each document containing term *i* and the centroid vector calculated without term *i*.

So, for example, if we have a collection of 10,000 documents. First we calculate a centroid vector which is the average of all 10,000 vectors. Next we calculate the sum of the similarities between each document and the centroid vector. This is the initial space density *D*. We also, for each document, store its individual contribution to *D*.

In the end, if *D* is large, it means that the documents were very similar to one another. However, if *D* is small, it means the documents were not very similar to one another.

Next, for each term we calculate a new space density without the term.

For example, for some term *i* we copy the initial space density to a variable D_i which will be the space density without term *i*.

If some term *i* occurs in ten documents, we find the ten values that those documents originally contributed to the initial space density and subtract them from D_i.

Now, for each of the ten documents, we calculate ten new similarities between the document vectors and the centroid vector but without any contribution from term *i*. These we add to D_i. The difference between *D* and D_i is the effect on the space density of term *i*.

If the D_i is less than *D*, it means that term *i* made the documents *more* like one another and is thus not a good discriminator. On the other hand, if D_i is larger than *D*, it means that term *i* was making the documents look less like one another and thus a good discriminator.

So, if (D_i – *D*) is negative, term *i* is a poor discriminator. If it is positive, term *i* is a good discriminator.

Complexity is on the order of $N+(DF)(w)(W)$ where *DF* is the average number of documents in which an average term occurs, *W* is the total number of words in the collection and *w* is the average number of terms per document vector.

This method increases the amount of space required in two ways. First, a vector of length N of original contributions is required.

Second, an inverted list giving, for each word, those documents containing the word is needed. The list will contain W entries and the amount of data stored will be proportional to the average number of documents in which an average word appears. Such an inverted file, however, is normally present at this stage of indexing in the form of the term-document matrix.

The advantage of using an inverted term-document matrix is that it quickly identifies those documents containing terms of interest rather than scanning, for each term, through the entire document-term matrix looking for documents that contain the term.

While the MCA method yields values that are only an approximation of the exact method, the values are very similar to the exact method while the savings in time to calculate the coefficients are very significant.

Crouch [Crouch 1988] reports that the MCA method was on the order of 527 times faster than the exact method on relatively small data sets. Larger data sets yield even greater savings as the time required for the exact method grows with the square of the number of documents while the MCA method grows linearly.

The basic MCA algorithm is given in Program 13. This program operates on a copy (*copy2*) of the database and runs in parallel with other routines. It uses unweighted (term counts only) copies of the *^doc* and *^index* matrices.

```
#!/usr/bin/mumps
# discrim.mps Feb 14, 2014
# Copyright 2014 Kevin C. O'Kane

      database "copy2"

      set D=^DocCount(1)  // number of documents
      set sq=0
      kill ^mca

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# calculate centroid vector ^c() for entire collection and
# the sum of the squares (needed in cos calc but should only be done once)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      for w=$order(^dict(w)) do
      . set ^c(w)=^dict(w)/D   // centroid is composed of avg word usage
      . set sq=^c(w)**2+sq     // The sum of the squares is needed below.

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Calculate total similarity of doc for all words (T) space by
# calculating the sum of the similarities of each document with the centroid.
# Remember and store contribution of each document in ^dc(dn).
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      set T=0
      for i=$order(^doc(i)) do
      . set x=0
      . set y=0

      . for w=$order(^doc(i,w)) do
      .. set d=^doc(i,w)
      .. set x=d*^c(w)+x                         // numerator of cos(c,doc) calc
      .. set y=d*d+y                             // part of denominator

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Calculate and store the cos(c,doc(i)).
# Remember in ^dc(i) the contribution that this document made to the total.
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      . if y=0 quit
      . set ^dc(i)=x/$zsqrt(sq*y)               // cos(c,doc(i))
      . set T=^dc(i)+T                           // sum the cosines
```

```
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# calculate similarity of doc space with words removed
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      for W=$order(^dict(W)) do

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# For each document containing W, calculate sum of the contribution
# of the cosines of these documents to the total (T).  ^dc(i) is
# the original contribution of doc i.  Sum of contributions is stored in T1.
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      . set T1=0,T2=0
      . for i=$order(^index(W,i)) do            // row of doc nbrs for word
      .. set T1=^dc(i)+T1                       // use prevsly calc'd cos

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# For each word in document i, recalculate cos(c,doc) but without word W
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      .. set x=0
      .. set y=0
      .. for w=$order(^doc(i,w)) do
      ... if w'=W do                          // if W not w
      .... set d=^doc(i,w)
      .... set x=d*^c(w)+x                   //  d*^c(w)+x
      .... set y=d**2+y

      .. if y=0 quit
      .. set T2=x/$zsqrt(sq*y)+T2               // T2 sums cosines without W

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# subtract original contribution with W (T1) and add contribution
# without W (T2) and calculate r - the change, and store in ^mca(W)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#     if old (T1) big and new (T2) small, density declines

      . set r=T2-T1*10000\1
      . write $j(r,6)," ",W,!

      use 5
```

```
    halt
```

Program 13 Modified centroid algorithm

In Program 13, the density of the space without word *i* is:

$$(T - T_{i1} + T_{i2})$$

where T is the original similarity between all documents and the centroid vector with no terms removed, T_{i1} is the sum of original similarity contributions to T of those documents in which term *i* appeared, and T_{i2} is the newly calculated sum of the contributions of those same documents without word *i*. Note that T_{i1} and T_{i2} only refer to the contribution to T by those documents which originally contained term *i*.

That is, we subtract the original contribution (T_{i1}) from T and then add in the replacement contribution T_{i2} calculated without the word under consideration.

The difference between the original density T and the density without word *i* is:

$$T - (T - T_{i1} + T_{i2})$$

which reduces to

$$(T_{i1} - T_{i2})$$

A larger value (higher space density) of either T_{i1} or T_{i2} means documents are more like one another while a smaller value (lower space density) means they are less like one another.

Thus, if T_{i1} is larger than T_{i2}, it means that for word *i*, removing it from the collection made the documents look less like one another. Consequently, term *i* makes documents look more like one another and it is a poor discriminator.

On the other hand, if T_{i1} is smaller than T_{i2}, it means that removing word *i* made the documents look more like one another and therefore term *i* is a good discriminator. Word *i* is a discriminator.

In order to give positive values to good discriminators and negative values to poor discriminators, in Program 13 we reverse the order of the subtraction to become:

$$(T_{i2} - T_{i1})$$

MCA discrimination co-efficients range from very large negative numbers (poor discriminators) to small positive numbers (good discriminators). Good discriminators are small positive values because they tend to be words used infrequently in few documents and, consequently, their effect on the space density is minimal. On the other hand, poor discriminators, by definition, occur in amny documents and thus their contribution to the document space density is large.

Because overall the numbers tend to be very small, we multiply the results by an arbitrary factor to bring them into a more easily managed range and (by the integer division operator \), we truncate fractional parts.

Figure 45 gives some sample good discriminators and Figure 46 gives some poor discriminators. The discrimination co-efficient is the first number, followed by the word followed by the inverse document frequency weight. In the good discriminator list, some technical words are not shown.

Note that the inverse document frequency weights often differ with discrimination co-efficients. For example, the words:

```
  982 phenytoin 6.66
  985 atenolol 7.01
  988 streptokinase 6.83
 1009 betacarotene 8.05
 1009 meningioma 6.89
```

are very near one another with regard to discrimination co-efficients (first column) but quite different with regard to inverse document frequency weights (third column).

Also note that in the list of poor discriminators, very high frequency and stop list words had already been eliminated at earlier stages and thus do not appear here. Also note that the discrimination co-efficients and the inverse document frequencies are in closer agreement.

Also, while in most texts words like *platelet, breast, glucose, laser, bladd[er], diabetic and cholesterol* would be important terms, in this specialized medical databse, they are not.

```
  956 restenosis 6.81
  965 valvuloplasty 6.52
  972 gerbil 8.05
  973 folate 7.11
  977 cpr 6.97
  980 sufentanil 7.49
  982 phenytoin 6.66
  985 atenolol 7.01
  988 streptokinase 6.83
 1009 betacarotene 8.05
 1009 meningioma 6.89
 1010 splenectomy 6.56
 1025 ribavirin 7.39
 1029 dobutamine 7.18
```

```
1031 hemangioma 6.78
1034 acetaldehyde 7.35
1056 lithium 6.89
1083 selenium 7.24
1083 taurine 7.56
1097 coarct 6.97
1098 digoxin 6.93
1116 hypoglycemia 6.81
1124 islet 6.78
1134 nalbuphine 8.05
1140 oxalate 7.21
1169 pseudoaneurysm 7.11
1187 methadone 7.64
1187 thymoma 7.24
1221 amiodarone 6.54
1226 mesothelioma 7.08
1230 propafenone 8.42
1236 surfactant 6.89
1299 pseudocyst 7.52
1305 ketanserin 7.42
1309 nicardipine 7.64
1312 pheochromocytoma 6.93
1319 rilmenidine 8.25
1322 sucralfate 7.13
1339 caffeine 6.69
1558 nabumetone 7.73
1875 tgfbeta 8.11
```

Figure 45 Good Discriminators

```
-107205 platelet 4.65
-86790 breast 4.54
-81255 glucose 4.55
-74248 laser 4.74
-70996 bladd 4.63
-62966 diabetic 4.77
-62359 cord 4.52
-61372 cholesterol 4.75
-59672 hiv 4.72
-57247 alcohol 4.63
-56692 cervical 4.51
-55539 arthrit 4.63
-52538 beta 4.61
-51880 thyroid 4.84
-51143 intake 4.52
-50648 diabete 4.52
-50302 hemorrhage 4.58
-48706 neuron 4.71
-48199 traine 4.71
-46889 image 4.59
-46835 fiber 4.68
-46793 motor 4.54
-46636 intestinal 4.55
-45818 arteries 4.51
-45737 repair 4.57
-45655 radiation 4.55
-45448 aneurysm 4.87
-43745 tachycardia 4.88
-43412 stroke 4.64
-43375 bypass 4.63
-41668 wound 4.69
-40662 carotid 4.81
-39584 urine 4.55
-39349 percutane 4.59
-39318 birth 4.60
-38682 burn 5.08
-38610 insulin 4.99
-38464 visu 4.63
-38450 leukemia 4.81
-38141 antagonist 4.52
-38110 fragment 4.52
-38006 output 4.53
-37868 food 4.73
```

```
-37465 energy 4.67
-37406 anesthesia 4.62
-37383 neck 4.62
```

Figure 46 Poor Discriminators

9.15 Calculating Inverse Document Weights (*idf.mps*)

The program *idf.mps* (see Program 14) calculates the *inverse document frequency* weights of the terms (stems) in the collection. These are the IDF weights of the terms in the collection *as a whole*. An example is shown in Figure 47.

Weights that fall below a command line supplied threshold are not stored in the *^idf(w)* vector.

The program *idf.mps* creates the vector *^idf(w)* where *w* is a vocabulary stem. The value stored at each element is the value of the IDF weight for the stem.

```
#!/usr/bin/mumps
# idf.mps
# Copyright 2014, 2022 Kevin C. O'Kane
# Feb 24, 2022

 zmain

 set doc=^DocCount(1)

 if '$data(%1) write "missing idfmin - using 4.0" set min=4.0
 else  set min=%1

 for w=$order(^df(w)) do
 . set x=$zlog2(doc/^df(w))
 . if x<min quit
 . set ^idf(w)=$justify(x,1,3)

 halt
```

Program 14 *idf.mps*

```
9.322 x-link
9.322 xenon
9.322 worthwhile
9.322 wish
9.322 wilcoxon
9.322 widene
9.322 went
9.322 wast
9.322 warn
9.322 wale
9.322 vulneare
9.322 vulgar
9.322 voltage-dependent
9.322 vincristine
9.322 vill
9.322 veteran
9.322 ventriculography
9.322 venom
9.322 vast
9.322 vasospasm
9.322 vancomycin
9.322 vagu
9.322 usage
9.322 urticaria
```

Figure 47 Example IDF Weights

9.16 Deletion of Low IDF Terms from the Collection (*idf-cutoff.mps*)

The program *idf-cutoff.mps* (see Program 15) deletes words from other system vectors if the IDF value for a word was not stored due to the value being below the system threshold.

The program also deletes document term vectors for documents where the number of terms remaining in the document falls below of command line supplied value. It also deletes rows from the term document matrix for terms whose IDF values were below the system threshold.

```
#!/usr/bin/mumps
# idf-cutoff.mps Feb 14, 2014
# Copyright 2014 Kevin C. O'Kane

# Feb 22, 2022

      zmain

      if '$data(%1) write "Missing parameter",! halt

      set wmin=%1
      set x=0

      open 3:"idf-deleted-words.txt,new"

      use 3

      for w=$order(^dict(w)) do
      . if $data(^idf(w)) quit
      . write w,!
      . kill ^idf(w) kill ^mca(w) kill ^dict(w)

      close 3
      use 5

      for d=$order(^doc(d)) do
      . for w=$order(^doc(d,w)) do
      .. if '$data(^idf(w)) kill ^doc(d,w),^index(w,d)

      open 3:"idf-deleted-documents.txt,new"

      use 3

      for d=$order(^doc(d)) do
      . set z=0
      . for y=$order(^doc(d,y)) set z=z+1
      . if z<wmin do
      .. for y=$o(^doc(d,y)) kill ^doc(d,y),^index(y,d)
      .. kill ^doc(d)
      .. write "Doc ",d," deleted count=",z,!
      .. set x=x+1

      close 3
      use 5
```

```
for w=$order(^index(w)) if $order(^index(w,""))="" kill ^index(w)

write !,?8,"IDF total docs deleted: ",x,!

set D=0
for d=$order(^doc(d)) set D=D+1
set ^DocCount(1)=D
write ?8,"IDF remaining documents: ",^DocCount(1),!

open 1:"DocCount,new" use 1 write D,! close 1 use 5

open 1:"idfWeights.tmp,new"

use 1

set wcount=0

for w="":$order(^idf(w)):"" do
. write ^idf(w)," ",w,!
. set wcount=wcount+1

use 5
close 1

write ?8,"IDF number of words remaining: ",wcount,!
halt
```

Program 15 *idf-cutoff.mps*

```
active
acute
addition
administer
analysis
assess
associate
blood
case
caus
cause
change
chronic
clinical
common
compare
complication
concentration
conclude
condition
control
correlate
data
```

Figure 48 *idf-deleted-words.txt* Example

9.17 Calculating Discrimination Coefficients (*discrim.mps*)

Discrimination coefficients are another way to determine the importance of a term in the collection. Discrimination coefficients [Willet 1985, Crouch 1988, Salton 1983] measure the ability of terms to differentiate one document from another. They are calculated based on the effect a term has on overall hyperspace density with and without a given term.

If the space density is *greater* after a term is removed, that means the term was making documents look less like one another (a good discriminator) while a term whose removal *decreases* the density, thus making documents look more like one another, is a poor discriminator.

The discrimination values for a set of terms in a given document collection are comparable to the values for the IDF weights but not exactly.

The basic procedure to calculate discrimination coefficients involves calculating the average of pair-wise similarities between all documents in the space. This is the documents space density.

Then, for each word, the average pair-wise similarities of all the documents is re-calculated without the word. The difference in the averages is the term discrimination value for the term in question.

In practice, this is a very expensive weight to calculate unless speed-up techniques are used.

The centroid algorithm approaches [Crouch 1988], are an attempt to improve the speed of calculation. The exact calculation, where all pairwise similarity values are calculated each time, is of complexity on the order of *(N)(N-1)(w)(W)* where *N* is the number of documents, *W* is the number of words in the collection and *w* is the average number of terms per document vector.

Crouch [Crouch 1988] discusses several methods to improve the speed of this calculation. The first of these, the centroid approximate algorithm, involves calculating the similarities of the documents with a centroid vector representing the collection as a whole rather than all pair-wise similarities.

In the centroid approximation, a centroid vector for the entire collection is first calculated. A centroid vector is the average of all the document vectors and, by analogy, represents a point at the center of the hyperspace.

When we use a centroid vector, rather than calculate all the pair-wise similarities, we only calculate the similarities between each document and the centroid vector. The average of these similarities is taken to be the space density which, although not exactly the same as the full pairwise scheme, has been shown to be very similar [Crouch 1987].

The prime advantage of this method is thus, rather than calculating N*N document-document similarities, we calculate only N similarities. This improves the complexity to be on the order of *(N)(w)(W)* which represents a significant improvement in performance.

A further modification, called the Modified Centroid Algorithm (MCA), is possible. It is based on the centroid approximation from above but further reduces the overall complexity.

In the MCA, we calculate an initial space density in the same manner as the centroid approximation from above. However, in the MCA approach, we retain and store the individual contribution of each document to the initial total document space density. This requires additional space on the order of **N** to store the original **N** contributions.

Subsequently, when calculating the effect of a some term *i*'s removal on the document space density, the algorithm first subtracts from the total space density the original contribution of each document that contains term *i* (using the stored values) and then it adds to the space density the similarity between each document containing term *i* and the centroid vector calculated without term *i*.

So, for example, if we have a collection of 10,000 documents. First we calculate a centroid vector which is the average of all 10,000 vectors. Next we calculate the sum of the similarities between each document and the centroid vector. This is the initial space density *D*. We also, for each document, store its individual contribution to *D*.

In the end, if *D* is large, it means that the documents were very similar to one another. However, if *D* is small, it means the documents were not very similar to one another.

Next, for each term we calculate a new space density without the term.

For example, for some term *i* we copy the initial space density to a variable D_i which will be the space density without term *i*.

If some term *i* occurs in ten documents, we find the ten values that those documents originally contributed to the initial space density and subtract them from D_i.

Now, for each of the ten documents, we calculate ten new similarities between the document vectors and the centroid vector but without any contribution from term *i*. These we add to D_i. The difference between *D* and D_i is the effect on the space density of term *i*.

If the D_i is less than *D*, it means that term *i* made the documents *more* like one another and is thus not a good discriminator. On the other hand, if D_i is larger than *D*, it means that term *i* was making the documents look less like one another and thus a good discriminator.

So, if (D_i – *D*) is negative, term *i* is a poor discriminator. If it is positive, term *i* is a good discriminator.

Complexity is on the order of *N+(DF)(w)(W)* where *DF* is the average number of documents in which an average term occurs, *W* is the total number of words in the collection and *w* is the average number of terms per document vector.

This method increases the amount of space required in two ways. First, a vector of length N of original contributions is required.

Second, an inverted list giving, for each word, those documents containing the word is needed. The list will contain W entries and the amount of data stored will be proportional to the average number of documents in which an average word appears. Such an inverted file, however, is normally present at this stage of indexing in the form of the term-document matrix.

The advantage of using an inverted term-document matrix is that it quickly identifies those documents containing terms of interest rather than scanning, for each term, through the entire document-term matrix looking for documents that contain the term.

While the MCA method yields values that are only an approximation of the exact method, the values are very similar to the exact method while the savings in time to calculate the coefficients are very significant.

Crouch [Crouch 1988] reports that the MCA method was on the order of 527 times faster than the exact method on relatively small data sets. Larger data sets yield even greater savings as the time required for the exact method grows with the square of the number of documents while the MCA method grows linearly.

The basic MCA algorithm is given in Program 16. This program operates on a copy (*copy2*) of the database and runs in parallel with other routines. It uses unweighted (term counts only) copies of the ^*doc* and ^*index* matrices.

```
#!/usr/bin/mumps
# discrim.mps Feb 14, 2014
# Copyright 2014 Kevin C. O'Kane

      database "copy2"

      set D=^DocCount(1)  // number of documents
      set sq=0
      kill ^mca

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# calculate centroid vector ^c() for entire collection and
# the sum of the squares (needed in cos calc but should only be done once)
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      for w=$order(^dict(w)) do
      . set ^c(w)=^dict(w)/D   // centroid is composed of avg word usage
      . set sq=^c(w)**2+sq     // The sum of the squares is needed below.

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Calculate total similarity of doc for all words (T) space by
# calculating the sum of the similarities of each document with the centroid.
# Remember and store contribution of each document in ^dc(dn).
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      set T=0
      for i=$order(^doc(i)) do
      . set x=0
      . set y=0

      . for w=$order(^doc(i,w)) do
      .. set d=^doc(i,w)
      .. set x=d*^c(w)+x                       // numerator of cos(c,doc) calc
      .. set y=d*d+y                           // part of denominator

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Calculate and store the cos(c,doc(i)).
# Remember in ^dc(i) the contribution that this document made to the total.
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      . if y=0 quit
      . set ^dc(i)=x/$zsqrt(sq*y)              // cos(c,doc(i))
      . set T=^dc(i)+T                         // sum the cosines
```

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# calculate similarity of doc space with words removed
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      for W=$order(^dict(W)) do

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# For each document containing W, calculate sum of the contribution
# of the cosines of these documents to the total (T).  ^dc(i) is
# the original contribution of doc i.  Sum of contributions is stored in T1.
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      . set T1=0,T2=0
      . for i=$order(^index(W,i)) do            // row of doc nbrs for word
      .. set T1=^dc(i)+T1                       // use prevsly calc'd cos

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# For each word in document i, recalculate cos(c,doc) but without word W
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      .. set x=0
      .. set y=0
      .. for w=$order(^doc(i,w)) do
      ... if w'=W do                          // if W not w
      .... set d=^doc(i,w)
      .... set x=d*^c(w)+x                   //  d*^c(w)+x
      .... set y=d**2+y

      .. if y=0 quit
      .. set T2=x/$zsqrt(sq*y)+T2               // T2 sums cosines without W

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# subtract original contribution with W (T1) and add contribution
# without W (T2) and calculate r - the change, and store in ^mca(W)
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#     if old (T1) big and new (T2) small, density declines

      . set r=T2-T1*10000\1
      . write $j(r,6)," ",W,!

      use 5
      halt
```

Program 16 Modified centroid algorithm

In Program 16, the density of the space without word *i* is:

$$(T-T_{i1}+T_{i2})$$

where T is the original similarity between all documents and the centroid vector with no terms removed, T_{i1} is the sum of original similarity contributions to T of those documents in which term *i* appeared, and T_{i2} is the newly calculated sum of the contributions of those same documents without word *i*. Note that T_{i1} and T_{i2} only refer to the contribution to T by those documents which originally contained term *i*.

That is, we subtract the original contribution (T_{i1}) from T and then add in the replacement contribution T_{i2} calculated without the word under consideration.

The difference between the original density T and the density without word *i* is:

$$T-(T-T_{i1}+T_{i2})$$

which reduces to

$$(T_{i1}-T_{i2})$$

A larger value (higher space density) of either T_{i1} or T_{i2} means documents are more like one another while a smaller value (lower space density) means they are less like one another.

Thus, if T_{i1} is larger than T_{i2}, it means that for word *i*, removing it from the collection made the documents look less like one another. Consequently, term *i* makes documents look more like one another and it is a poor discriminator.

On the other hand, if T_{i1} is smaller than T_{i2}, it means that removing word *i* made the documents look more like one another and therefore term *i* is a good discriminator. Word *i* is a discriminator.

In order to give positive values to good discriminators and negative values to poor discriminators, in Program 16 we reverse the order of the subtraction to become:

$$(T_{i2} - T_{i1})$$

MCA discrimination co-efficients range from very large negative numbers (poor discriminators) to small positive numbers (good discriminators). Good discriminators are small positive values because they tend to be words used infrequently in few documents and, consequently, their effect on the space density is minimal. On the other hand, poor discriminators, by definition, occur in amny documents and thus their contribution to the document space density is large.

Because overall the numbers tend to be very small, we multiply the results by an arbitrary factor to bring them into a more easily managed range and (by the integer division operator \), we truncate fractional parts.

Figure 1 gives some sample good discriminators and Figure 2 gives some poor discriminators. The discrimination co-efficient is the first number, followed by the word followed by the inverse document frequency weight. In the good discriminator list, some technical words are not shown.

Note that the inverse document frequency weights often differ with discrimination coefficients. For example, the words in Figure 49

```
  982 phenytoin 6.66
  985 atenolol 7.01
  988 streptokinase 6.83
 1009 betacarotene 8.05
 1009 meningioma 6.89
```

Figure 49 IDF vs. MCA Weights

are very near one another with regard to discrimination co-efficients (first column) but quite different with regard to inverse document frequency weights (third column).

Also note that in the list of poor discriminators, very high frequency and stop list words had already been eliminated at earlier stages and thus do not appear here. Also note that the discrimination coefficients and the inverse document frequencies are in closer agreement.

Also, while in most texts words like *platelet, breast, glucose, laser, bladd[er], diabetic and cholesterol* would be important terms, in this specialized medical databse, they are not.

```
 956 restenosis 6.81
 965 valvuloplasty 6.52
 972 gerbil 8.05
 973 folate 7.11
 977 cpr 6.97
 980 sufentanil 7.49
 982 phenytoin 6.66
 985 atenolol 7.01
 988 streptokinase 6.83
1009 betacarotene 8.05
1009 meningioma 6.89
1010 splenectomy 6.56
1025 ribavirin 7.39
1029 dobutamine 7.18
1031 hemangioma 6.78
1034 acetaldehyde 7.35
1056 lithium 6.89
1083 selenium 7.24
1083 taurine 7.56
1097 coarct 6.97
1098 digoxin 6.93
1116 hypoglycemia 6.81
1124 islet 6.78
1134 nalbuphine 8.05
1140 oxalate 7.21
1169 pseudoaneurysm 7.11
1187 methadone 7.64
1187 thymoma 7.24
1221 amiodarone 6.54
1226 mesothelioma 7.08
1230 propafenone 8.42
1236 surfactant 6.89
1299 pseudocyst 7.52
1305 ketanserin 7.42
1309 nicardipine 7.64
1312 pheochromocytoma 6.93
1319 rilmenidine 8.25
1322 sucralfate 7.13
1339 caffeine 6.69
1558 nabumetone 7.73
1875 tgfbeta 8.11
```

Figure 50 Good Discriminators

```
-107205 platelet 4.65
-86790 breast 4.54
-81255 glucose 4.55
-74248 laser 4.74
-70996 bladd 4.63
-62966 diabetic 4.77
-62359 cord 4.52
-61372 cholesterol 4.75
-59672 hiv 4.72
-57247 alcohol 4.63
-56692 cervical 4.51
-55539 arthrit 4.63
-52538 beta 4.61
-51880 thyroid 4.84
-51143 intake 4.52
-50648 diabete 4.52
-50302 hemorrhage 4.58
-48706 neuron 4.71
-48199 traine 4.71
-46889 image 4.59
-46835 fiber 4.68
-46793 motor 4.54
-46636 intestinal 4.55
-45818 arteries 4.51
-45737 repair 4.57
-45655 radiation 4.55
-45448 aneurysm 4.87
-43745 tachycardia 4.88
-43412 stroke 4.64
-43375 bypass 4.63
-41668 wound 4.69
-40662 carotid 4.81
-39584 urine 4.55
-39349 percutane 4.59
-39318 birth 4.60
-38682 burn 5.08
-38610 insulin 4.99
-38464 visu 4.63
-38450 leukemia 4.81
-38141 antagonist 4.52
-38110 fragment 4.52
-38006 output 4.53
-37868 food 4.73
-37465 energy 4.67
```

```
-37406 anesthesia 4.62
-37383 neck 4.62
```

Figure 51 Poor Discriminators

```
-49132 children 4.052
-47930 tumor 4.082
-33525 cent 4.536
-32843 plasma 4.030
-27669 flow 4.090
-27545 acid 4.210
-27423 rats 4.341
-27249 ventricular 4.331
-27096 muscle 4.278
-27030 women 4.169
-26944 receptor 4.454
-23540 week 4.000
-21105 lung 4.567
…
   668 gallstone 8.837
   672 vaccine 7.737
   708 gaba 8.474
   710 splenectomy 8.185
   730 verapamil 8.185
   731 hear 7.737
   748 stent 9.059
   759 vitamin 6.837
   779 igf-i 9.059
   796 magnesium 8.059
   803 ptca 8.644
   825 prostatic 7.474
   831 seizure 6.786
   839 mast 7.944
   854 colorectal 7.737
   858 reservoir 7.474
   905 polyp 9.059
   919 oesophageal 8.185
   920 sucralfate 8.837
  1161 morphine 8.059
  1273 cocaine 8.322
```

Figure 52 Sample Discrimination Coefficients[19]

19 Format is: discrimination coefficient, stem, IDF weight.

9.18 Weighting Terms in Document Vectors (*weight.mps*)

Program 17 converts the numeric word counts in the document term and term document matrices to weights by multiplying the count for each term by its weight in the collection as a whole. In this case, we use the values in the vector ^*idf(w)* although ^*mca(w)* could be used instead.

Words whose weights are below a command line threshold are discarded.

Figure 53 gives a sample of the revised document term vectors and the newly calculated weights of the terms in the document. Figure 54 gives a sample of the term document matrix with the weights of the terms in the documents shown.

```
#!/usr/bin/mumps
# weight.mps

# Feb 23, 2022

      zmain

 if '$d(%1) set m1=10.
 else  set m1=%1

 if '$d(%2) set m2=10.
 else  set m2=%2

 kill ^index

 set delwords=0
 set deldocs=0

 open 2:"document-term-matrix-weighted.txt,new"

 for d=$order(^doc(d)) do
 . use 2 write !,"doc=",d,?15
 . use 2 write ^title(d),!,?15
 . for w=$order(^doc(d,w)) do
 .. set x=^idf(w)*^doc(d,w)
 .. if x<m1 kill ^doc(d,w) set delwords=delwords+1 quit
 .. set ^doc(d,w)=x
 .. set ^index(w,d)=x
 .. write w,"(",x,") "
 . write !
 close 2

 for d=$o(^doc(d)) do
 . set i=0
 . for w=$o(^doc(d,w)) do
 .. set i=i+1
 . if i<m2 kill ^doc(d) set deldocs=deldocs+1

 open 2:"term-document-matrix-weighted.txt,new"
 use 2
 for w=$order(^index(w)) do
 . write w,?26
 . for d=$order(^index(w,d)) do
 .. write d,"(",^index(w,d),") "
```

```
. write !

use 5
close 2

write ?8,delwords," document word instances with weight < ",m1," deleted ",!
write ?8,deldocs," docs with less than ",m2," words deleted",!!
```

Program 17 *weight.mps*

```
doc=1          The binding of acetaldehyde to the active site of ribonuclease:
               absence(24.12) adduct(17.674) amino(28) aris(14.948)
               catalytic(17.288) diminish(12.718) formation(10.21)
               inhibition(10.03) peak(10.18) phosphate(66.44) protect(11.334)
               residue(30.228)

doc=100        Edrophonium provocation test in the diagnosis of diffuse
               baseline(11.674) diagnose(10.148) diffuse(12.948)
               manometry(49.932) oesophageal(24.555) positive(17.008)
               provocate(44.185) spasm(17.674) typical(13.114)

doc=1000       Cooperation between CD16(Leu-11b)+ NK cells and HLA-DR+ cells in
               abil(10) cell-mediate(27.177) fibroblast(13.888) lysi(57.295)
               nature(10.948) supernatant(16.948) target(41.334) viru(12.504)

doc=1001       In vitro production of cholera toxin-like activity by Plesiomonas
               diarrhea(15.888) elongation(18.644) strain(26.228)

doc=1002       Invasive disease due to Streptococcus pneumoniae in an area with
               area(12.222) invasive(13.674) isolate(14.948) meningit(18.118)
               penicillin(18.118) person(22.948) pneumococcal(18.644)
               pneumoniae(17.288) streptococcus(16.948)

doc=1003       Enzymatic modification of glycocalyx in the treatment of
               endocardit(16.118) experiment(12.411) infect(18.184)
               penicillin(63.413) sterilization(18.118)

doc=1004       Virulence studies, in mice, of transposon-induced mutants of
               aureus(22.422) capsule(24.555) mice(18.27) mutant(34.576)
               staphylococcus(14.792) strain(78.684)

doc=1005       Isolation of a 220-kilodalton protein with lectin properties from
               antibody(14.628) attachment(16.948) kilodalton(17.674)
               lectin(25.422) property(16.608) purify(11.832) strain(13.114)

doc=1006       Antigenic relationships between human caliciviruses and Norwalk
               antigen(12) distinct(11.244) ident(11.572) japan(27.966)
               strain(26.228) viru(31.26) virus(17.288)
```

Figure 53 Weighted Document Term Matrix[20]

20 Long titles truncated. Numbers shown are the weights of rthe terms in the document based on IDF weights.

abdomen	3109(15.888) 4344(15.888) 771(23.832) 976(15.888)
abdominal	1179(11.244) 1345(22.488) 1860(11.244) 1875(16.866) 2140(11.244) 2213(11.244) 2214(11.244) 2496(22.488) 2810(16.866) 3109(11.244) 3178(11.244) 3255(11.244) 3256(11.244) 3405(11.244) 3676(11.244) 3887(11.244) 4277(22.488) 4279(11.244) 748(11.244)
abduct	1923(26.511) 312(26.511) 781(17.674)
aberrant	2232(25.932) 4321(17.288)
abil	1000(10) 1275(10) 3055(10) 312(10) 3300(10) 3726(10) 3860(10) 4014(15) 4036(15) 4195(10) 4392(10) 4463(10) 4897(10) 4902(10) 524(20) 807(15)
ablation	3948(34.576) 4419(17.288)
able	26(12.06) 4813(30.15)
abnorm	2121(18.861) 2291(37.722) 2595(12.574) 3079(12.574) 3208(12.574) 3280(12.574) 3281(12.574) 3303(12.574) 93(18.861) 934(18.861)
abolish	2520(13.2) 269(13.2) 4225(13.2) 4331(13.2) 507(19.8) 747(13.2)
abortion	1458(25.422) 1870(42.37) 3722(25.422) 93(25.422)

Figure 54 Weighted Term Document Matrix[21]

21 Format is document number followed by weight of term in document in parentheses based in IDF weights.

10 Thesaurus Construction

10.1 Construction of Term Phrases

Recall (at the expense of precision)can be improved if additional related terms are added to a query. Thus, a query for *antenna* will usually result in more hits in the database if the related term *aerial* is added.

An increase in recall, however, is often accompanied by a decrease in precision. As is evidenced by the fact that while *aerial* is a commonly used synonym for *antenna*, as in *television aerial*, it can also refer to a dance move, a martial arts move, skiing, various musical groups, performances, and any activity that is done at a height (*e.g.*, aerial photography). Thus, adding it to a query with *antenna* has the potential to introduce many extraneous hits.

Identification of phrases, however, has the potential to increase precision. Phrases can be regarded as composite terms of high specificity - such as *television aerial* noted above. While both *television* and *aerial* individually are broad terms, the phrases *television aerial* and *television antenna* are quite specific.

When a phrase is identified, it should be added as a single term in the document vector (see below).

Phrases can be identified by both syntactic and statistical methods.

It is possible to discover connections between vocabulary terms based on their frequency of co-occurrence. Terms that co-occur frequently together are likely to be related and this can indicate that the words may be synonyms used to express a similar concepts or phrase components

For example, the strong statistical relationship such as between the words *artificial* and *intelligence* in a computer science database is due to the phrase *artificial intelligence* which names a branch of computing. In this case, the relationship is not that of a synonym but that of a phrase. Similarly, in a medical data base, terms such as *circadian rhythm* and *vena cava* and *herpes simplex* are also concepts expressed using more than one term.

On the other hand, terms such as *synergism* and *synergistic*, *cyst* and *cystic*, *schizophrenia* and *schizophrenic*, *nasal* and *nose*, and *laryngeal* and *larynx* are examples of synonym relationships.

In other cases, the relationship is not so tight so as to be a full synonym but it may express a categorical relationship such as *anesthetic* and *halothane; analgesia* and *morphine*, *nitrogen* and *urea;* and *nurse* and *personnel*.

Regardless of the relationship, a basic thesaurus, that is, a data structure that lists words in groups of synonyms and related concepts, can be used to:

1. Augment queries with related words to improve recall. For example, if the user enters the term *halothane,* the system might add, at a lower weight, the term *anesthetic* to the query. This would broaden the query thus improving recall but at the expense of precision.

2. Combine multiple, related, infrequently occurring terms into broader, more frequently occurring terms. In many systems, low frequency terms are discarded. For example, a collection might contain a number of specific medication names all related to the same medical condition which individually, occur infrequently (*e.g.*, names of medicines associated with treatment of high blood pressure, of which there are many). However, by gathering these terms into a collective term, their aggregate frequency of occurrence might be high enough to be included in the indexing vocabulary. In place of the original terms in the documents, a stand-in token for the category would replace each original word instance.
3. Create middle frequency composite terms from otherwise unusable, high frequency terms. While some words individually may have frequencies of occurrence too high to be useful as index terms (*e.g.,* computer), taken as a component of a phrase with other high frequency terms (*e.g.,* computer aided instruction), the phrase's frequency may drop below the high frequency cutoff.

To build a thesaurus we first we construct a square term-term correlation matrix which gives the frequency of co-occurrence of terms with one another. A term-term matrix is a symmetric[22] square matrix with terms denoting the rows and columns.

The values in the cells of the matrix indicate the number of times a a row term and a column term co-occur. The diagonal is the number of times a term occurs. Only the lower or upper triangular matrix need be computer since they are equivalent (the number of times some term A occurs with term B is the same as term B with term A). A brief example is given in Figure 55.

	birds	cats	dogs	fis h
birds	5	3	6	4
cats	3	4	7	3
dogs	6	7	8	3
fish	4	3	3	9

Figure 55 Term-Term Matrix

To build a term-term matrix we inspect the documents for words that co-occur with one another. If some term A occurs in 20 documents and if term B also occurs in these same documents, the term-term correlation matrix cell for row A and column B will have a value of 20. Since the relationship between term A and B is always the same as the relationship between term B and A, a term-term correlation matrix's lower diagonal matrix is the same as the upper diagonal matrix. The diagonal itself is the number of times a term occurs with itself which is the count of the number of documents in which the term occurs.

22 The matrix is equal to its transpose.

Calculating a complete term-term correlation matrix based on all documents in a large collection can be very time consuming. In many cases, a term-term matrix potentially contains many billions of elements (the square of the number of vocabulary terms) summed over the entire collection of documents. In practice, however, it is only necessary to sample a representative part of the total collection. That is, you can calculate a representative matrix by looking at every fifth, tenth or twentieth document, *etc*., depending on the total size of the collection.

However, be aware that many collections may contain clusters of documents concerning specific topics. For example, a number of documents taken from the same sub-specialty journal may appear consecutively in the collection. As these would deal with a particular aspect of a discipline, the word usage might be skewed. Thus, it is better to sample across all the documents than to only process documents in a initial fraction of the collection.

As many words never occur with others, especially in specialized technical collections such as the OHSU medical data base, a term-term matrix for a vocabulary rich technical discipline may be very sparse[23]. More general topic collections, however, will probably have matrices that are less sparse.

10.2 Building a Term-Term Co-Occurrence Matrix

A *term-term co-occurrence matrix,* also referred to as the *term connection matrix*, is an *N x N* square matrix, where *N* is the number of terms in the vocabulary. Its elements give the number of documents common to each pair of terms. From a formal point of view, it is:

$$TermTerm = DocTerm^T \cdot DocTerm$$

That is, the product of the document-term matrix *DocTerm* with its transpose.

In our *TermTerm* matrix, the rows and columns are term identifiers and the cells contain a count of the number of documents which contained both terms.

Because the number of times a term occurs in a document is not important in terms of calculating term co-occurrences, for purposes of calculation, we treat the elements of *DocTerm* as binary values where a 0 indicates the term is not present in the document and a non-zero value indicates the term is present.

The elements of the *TermTerm* matrix tt_{jk} for terms *j* and *k* can be calculated from the document-term matrix elements d_{ij} for document *i* and terms *j* and *k* where *i* ranges from 1 to *N* the total number of document as follows:

$$tt_{jk} = d_{1j} d_{1k} + d_{2j} d_{2k} + d_{3j} d_{3k} \ldots . d_{Nj} d_{Nk}$$

23 For example, pediatric terminology is unlikely to be common in articles about adult nutrition.

In other words, the similarity between term i and term j is the sum of the number of times they co-occur in a document.

For example, is you have 3 three documents entitled:

Doc_1 = Pseudomonas-aeruginosa cystic-fibrosis.

Doc_2 = Pseudomonas-aeruginosa immune response

Doc_3 = Immune complexes in cystic-fibrosis

Altogether we have five significant terms: *pseudomonas-aeruginosa, immune, cystic-fibrosis, response,* and *complexes* which are column headings for the matrix *DocTerm* (rows here are document numbers) and the row headings for $DocTerm^T$ (column headings are documents). Thus row one of *DocTerm* indicates that the document only contained the terms *pseudomonas-aeruginosa* and *cystic-fibrosis*. The first row of the $DocTerm^T$ matrix indicates that the term *pseudomonas-aeruginosa* occurs in documents 1 and 2 but not document 3.

In the *TermTerm* matrix, both row and column headings are terms. The first row indicates that the term *pseudomonas-aeruginosa* occurs twice in the collection as a whole (diagonal element), once each with the terms *immune, cystic-fibrosis, and response,* but not at all with the term *complexes.*

$$DocTerm=\begin{pmatrix}10100\\11010\\01101\end{pmatrix} \qquad DocTerm^T=\begin{pmatrix}110\\011\\101\\010\\001\end{pmatrix}$$

Consequently, the Term-Term matrix will thus be:

$$TermTerm = DocTerm^T \cdot DocTerm = \begin{pmatrix} 2\,1\,1\,1\,0 \\ 1\,2\,1\,1\,1 \\ 1\,1\,2\,0\,1 \\ 1\,1\,0\,1\,0 \\ 0\,1\,1\,0\,1 \end{pmatrix}$$

A Mumps program to calculate a term-term matrix is given in Program 18. In this figure, the global array ^*doc* is assumed to be a modified document-term matrix similar to the ones used in other examples except that the values of the rows are either 0 or 1, not counts or weights. A 1 indicates that a term is present and a 0 indicates it is not.

```
#!/usr/bin/mumps
# Copyright 2014, 2022 Kevin C. O'Kane
# tt1.mps Feb 15, 2022

     zmain

 if '$d(%1) set min=0.25
 else  set min=%1

 if '$d(%2) set min1=5
 else  set min1=%2

 open 1:"tt.tmp,new"

 use 1
 for d=$order(^doc(d)) do
 . for w=$order(^doc(d,w)) do
 .. for w1=w:$order(^doc(d,w1)):"" do
 ... if w1=w quit
 ... write w," ",w1,!

 close 1

 shell sort < tt.tmp | uniq -c | sort -nr -k 1 > tt.sorted.tmp

 open 1:"tt.sorted.tmp,old"
 open 2:"tt.tmp,new"

# input format: #co-occur w1 w2
# output format: relationship w1 w1 #co-occur ^df(w1) ^df(w2)

 kill ^tt

# input format: co-occur w1 w2
# out format: wgt w1 w2 co-occur df(w1) df(w2)

 for  do
 . use 1
 . read a
 . if '$test break
 . set a=$zblanks(a)
 . set w1=$piece(a," ",2)
 . set w2=$piece(a," ",3)
 . set co=+a
 . if co<min1 quit
 . set c1=co/(^df(w1)+^df(w2)-co)
 . if c1<min quit
 . set c1=$j(c1,1,3)
 . set ^tt(w1,w2)=c1
 . set ^tt(w2,w1)=c1
 . use 2
 . write ^tt(w1,w2)," ",w1," ",w2," ",co," ",^df(w1)," ",^df(w2),!

 close 1,2
 use 5
 shell sort -n < tt.tmp > term-term-matrix-sorted.txt
```

Program 18 *tt1.mps*

```
0.250 angioplasty ptca 7 27 8
0.250 cili iris 3 7 8
0.250 deferoxamine iron 6 8 22
0.250 deletion duchenne 3 9 6
0.250 doxorubicin mg/m 3 7 8
0.250 epiderm keratinocyte 4 11 9
0.250 esophagit sucralfate 3 8 7
0.250 magnetic resonance 22 54 56
0.250 melanocytic nevi 3 8 7
0.273 angioplasty transluminal 9 27 15
0.273 calculi lithotripsy 3 7 7
0.286 disc discectomy 4 12 6
0.286 enflurane isoflurane 4 7 11
0.292 cava vena 7 14 17
0.300 grave hyperthyroidism 3 8 5
0.304 colit ulcerative 7 15 15
0.308 duchenne dystrophy 4 6 11
0.308 fertilization fertilize 4 11 6
0.333 cleave fertilize 3 6 6
0.333 extracorporeal lithotripsy 5 13 7
0.333 fertilize oocyte 4 6 10
0.364 nitr oxide 4 7 8
0.400 deferoxamine hydroxyl 4 8 6
0.400 purpura thrombocytopenic 4 7 7
0.409 adenylate cyclase 9 15 16
0.429 cardiomyopathy hypertrophic 6 12 8
0.500 fertilization oocyte 7 11 10
0.600 erythematosus lupu 9 11 13
0.625 igf-i insulin-like 5 6 7
```

Figure 56 term-term-matrix-sorted.txt

Salton suggested two metrics for term-term association.

In the first formula [Salton 1983], which was used in Program 18, is shown below using Salton's original notation. In Salton's original, the $t_{i,j}$ are the weights of term *j* in document *i*. In Program 18, the values of t_{ij} are either zero (indicating that term *j* did not occur in document *i* or one, indicating that it did. The weight of a term is not very relevant when computing co-occurences. Either the terms do co-occur or they do not.

$$\text{SIMILAR}(TERM_k, TERM_h) = \frac{\sum_{i=1}^{n} t_{i,k} t_{i,h}}{\sum_{i=1}^{n} (t_{i,k})^2 + \sum_{i=i}^{n} (t_{i,h})^2 - \sum_{i=1}^{n} t_{i,k} t_{i,h}}$$

If the $t_{i,j}$ are taken to be counts, the figure:

$$\sum_{i=1}^{n} (t_{i,k})^2 + \sum_{i=i}^{n} (t_{i,h})^2$$

becomes the product of the document frequencies of terms k and h as used in Program 18 because when the t_{ik} either zero or one

$$\sum_{i=1}^{n} (t_{i,k})^2$$

becomes the number of documents that term k appeared in. Similarly,

$$\sum_{i=1}^{n} t_{i,k} t_{i,h}$$

is the count of the number of the co-occurrences between terms k and h.

Salton also proposed a second formula [Salton 1988] the $d_{i,j}$ are the weight of term j in document i:

$$\mathrm{Sim}(T_j, T_k) = \frac{\sum_{i=1}^{n} (d_{i,j})^2}{\sqrt{(\sum_{i=1}^{n} (d_{i,j})^2 \cdot \sum_{i=1}^{n} (d_{i,k})^2)}}$$

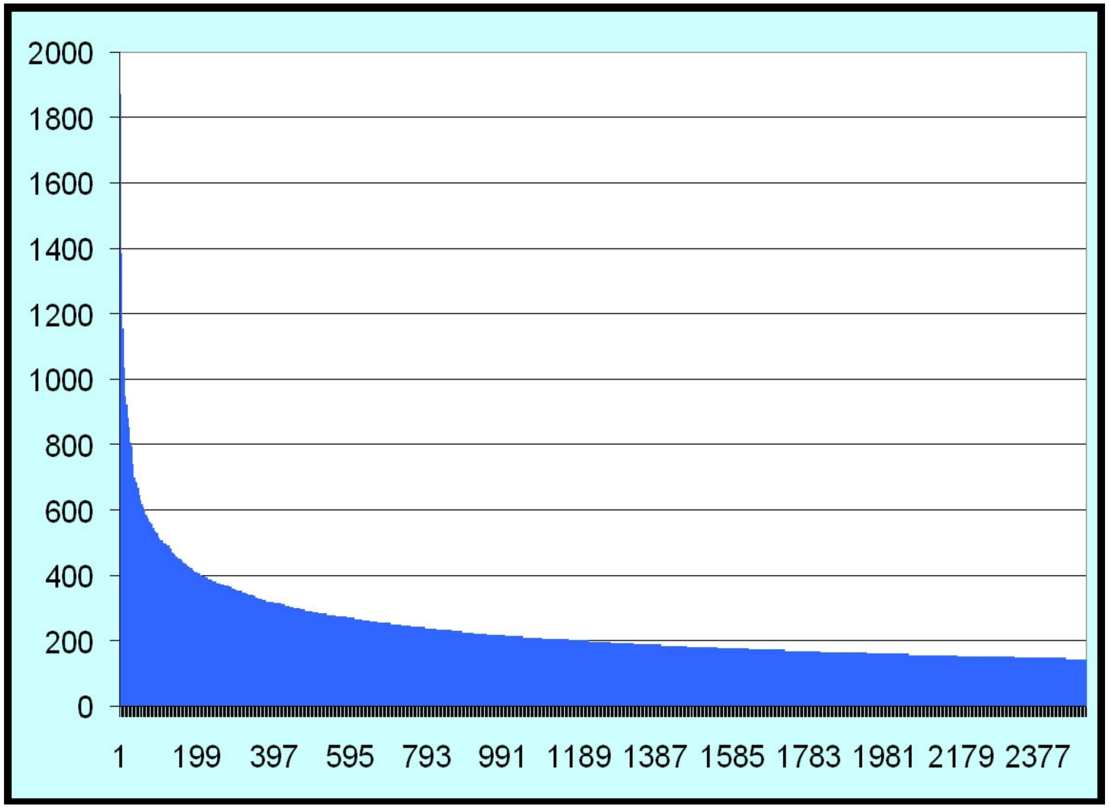


Figure 57 Frequency of term co-occurrences

In Figure 57, the frequency of co-occurrence (number of times two words co-occur together, vertical axis) for the entire OHSU collection is plotted against frequency of co-occurrence rank (horizontal axis). With regard to the horizontal axis, the term pair that co-occur the most frequently (a bit less than 1200 times) is rank number one (appears first), the term pair co-occurring next most frequently next, and so on.

As can be seen, the frequency of co-occurrence drops off rapidly to a nearly constant. Thus, only a few term pairs in this vocabulary, roughly the top 300, stand out as significantly more likely to co-occur than the remainder of the possible combinations due to chance alone.

10.3 Proximity Based Correlations

10.3.1 Term Proximity

Another modification to improve the detection of term-term relationships is shown in Program 19. It involves retaining, during initial document scanning, the relative positions of each term in each document in the collection. Subsequently, when calculating the term-term matrix, term proximity can be taken into account.

The assumption is that terms that co-occur frequently near one another an predominately in the same order are in fact closely related.

Recall[24] that when we initially scanned the documents *^doc()* stored the position of each term relative to the start of the document in the third index:

```
^doc(DocNbr,Word,Position)
```

The value for *Position* indicates if the term is the first, second, third, and so forth term in the document. Thus, this records, for each term in each document, the position of each instance of each *word* relative to the beginning of the document.

With this information, it becomes possible to calculate the distance between co-occurring terms as well as note which term appears before the other. Terms that frequently co-occur close to one another and normally in the same order are more likely to be related than terms that co-occur at a distance and in no preferred order with respect to one another.

A revised term-term matrix calculation taking into account proximity and order is calculated as follows:

1. for each document *k*
2. for each term *i* in *k*
3. for each other term *j* in *k* where term *j* is alphabetically greater than term *i*
4. for each position *m* of term *i*

24 Line 25, Program Error: Reference source not found, page Error: Reference source not found.

5. for each position *n* of term *j*
6. calculate a weight based on the distance between the terms and add this to the global array *^ttp(i,j)*
7. add to global array *^ttp(i,j,1)* the signed difference between the terms *n-m*. That is, the distance in word units between the two terms.

The weight added to *^ttp(i,j)* in step 6 is calculated with the formula:

```
set dd=$zlog(1/$zabs(m-n)*20+1)\1
```

which yields, for various distances as shown in Figure 58.

Thus, in Program 19, terms immediately next to one another receive a score of 3 while terms more than eleven positions apart receive a score of zero.

For each pair *{i,j}* the third level index is the sum of signed distances between *m* and n. After all co-occurrences have been summed, a positive or negative value for this term indicates a preference as to which term appears first most often. Values of zero or near zero indicate that the terms appear in no specific order relative to one another.

In the script file *index.script t*he program *proximity.mps* works on a copy of the database so that it may be executed in parallel. It's output (to *stdout*) is subsequently sorted into a final result file.

```
$zabs(m-n)=1  result 3
$zabs(m-n)=2  result 2
$zabs(m-n)=3  result 2
$zabs(m-n)=4  result 1
$zabs(m-n)=5  result 1
$zabs(m-n)=6  result 1
$zabs(m-n)=7  result 1
$zabs(m-n)=8  result 1
$zabs(m-n)=9  result 1
$zabs(m-n)=10 result 1
$zabs(m-n)=11 result 1
$zabs(m-n)=12 result 0
$zabs(m-n)=13 result 0
$zabs(m-n)=14 result 0
$zabs(m-n)=15 result 0
```

Figure 58 Proximity calculations

```
#!/usr/bin/mumps
# ~/Medline2012/proximity.mps
# Copyright 2014, 2022 Kevin C. O'Kane

# Mar 20, 2022

     zmain

# ttx term-term correlation matrix
# calculate term-term proximity coefficients within env words

 if '$data(%1) write "Missing parameter",! halt

 kill ^ttp  //* delete any old term-term correlation matrix

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# for each document k, sum the co-occurrences of words i and j
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


# for each document k

 set k=""
 for  do
 . set k=$order(^doc(k))
 . if k="" break

# for each term i in p k

 . set i=""
 . for  do
 .. set i=$order(^doc(k,i))
 .. if i="" break

# for each other term j in doc k

 .. set j=i
 .. for  do
 ... set j=$order(^doc(k,j))
 ... if j="" break

# for each position m of term i in doc k
 ... set m=""
 ... for  do
 .... set m=$order(^doc(k,i,m))
 .... if m="" break

# for each position n of term j in doc k
```

```

.... set n=""
.... for  do
..... set n=$order(^doc(k,j,n))
..... if n="" break

# calculate and store weight based on proximity

..... if $zabs(m-n)<5 quit
..... set dd=$zlog(1/$zabs(m-n)*20+1)\1
..... if dd<1 quit
..... if '$data(^ttp(i,j)) set ^ttp(i,j)=dd,^ttp(i,j,1)=n-m
..... else  set ^ttp(i,j)=^ttp(i,j)+dd,^ttp(i,j,1)=^ttp(i,j,1)+(n-m)

open 1:"term-term-proximity-similarities.tmp,new"
use 1

set i=""
for  do
. set i=$order(^ttp(i))
. if i="" break
. set j=""
. for  do
.. set j=$order(^ttp(i,j))
.. if j="" break
.. if ^ttp(i,j)<%1 kill ^ttp(i,j) quit
.. write ^ttp(i,j)," ",i," ",j," ",^ttp(i,j,1),!

close 1
```

Program 19 *proximity.mps*

Figure 59 contains some of the top ranking proximity based term-term proximity based correlation calculation from Program 19. All terms are shown in stem form.

In Figure 59 the first column gives the sum of the proximity weight scores while the last column gives net difference in the positions. The terms on each line are presented in alphabetic order. However the last column indicates which term tends to precede which. A large positive number favors the order shown while a large negative number favors the reverse order. A number near zero indicates that the terms are likely to appear in either order.

1721	infarct myocardial	-465
1445	dens lipoprotein	73
1184	abstract word	1184
1128	magnetic resonance	376
1095	cord spin	-309
1057	lymph node	313
964	arthrit rheumatoid	-307
936	female male	-43
878	cholesterol lipoprotein	43
872	blind double	-304
859	compute tomography	255
811	imag resonance	-189
762	carry out	262
742	determine whether	321
733	cholesterol dens	-137
686	mitr valve	114
660	state unit	-240
649	centre nerv	221
639	amino sequence	-7
638	aortic valve	194
633	guinea pig	32
629	care health	-97
625	imag magnetic	-439
610	cystic fibrosis	179
587	head neck	149
566	death sudden	-191
563	biopsy specimen	230
558	electron microscopy	111
548	marrow transplant	113
544	tract urin	-156
544	diabete mellitus	161
531	erratum publish	-177
530	sensit specificity	210
530	morbid mort	110
522	resistance vascular	-151
522	excretion urin	-283
522	ejection fraction	130
519	cerebrospinal fluid	106
517	coli escherichia	-246
513	diastolic end	-58
508	man old	-207
500	anti monoclon	-97
475	state steady	-174
470	erythematosus lupu	-134
469	randomize trial	143
466	diastolic systolic	-58

Figure 59 Ranked Proximity Correlations

A proximity based term-term matrix can yield substantially better results although it is considerably more expensive to calculate in terms of time and space. The example was calculated on the first 10,000 documents of the data base. Again, it should be noted, that uncovering term-term relationships need not involve analysis of the entire collection, just a representative sample of it.

```
#!/usr/bin/mumps
# ~/Medline2012/proxLoad.mps
# Copyright 2014, 2022 Kevin C. O'Kane

# Feb 15, 2022

      zmain

      open 1:"phrases.txt,new"

# input format: #co-occur w1 w2 direction

      for  do
      . use 5
      . read a
      . if '$test break
      . set wgt=$piece(a," ",1)
      . set w1=$piece(a," ",2)
      . set w2=$piece(a," ",3)
      . set dir=$piece(a," ",4)
      . set s=$zlog(wgt)
      . use 1
      . for d=$order(^index(w1,d)) do
      .. if $data(^index(w2,d)) do
      ... set w1stem=w1
      ... set w2stem=w2
      ... if $data(^stem(w1)) set w1=$order(^stem(w1,""))
      ... if $data(^stem(w2)) set w2=$order(^stem(w2,""))
      ... if dir<0 set prs=w2_"-"_w1
      ... else  set prs=w1_"-"_w2
      ... write d," ",s," ",prs,!
      ... set ^dict(prs)=wgt
      ... set ^df(prs)=wgt
      ... set ^phrase(w1stem,prs)=""
      ... set ^phrase(w2stem,prs)=""
      ... set ^doc(d,prs)=s
      ... set ^idf(prs)=s
      ... set ^index(prs,d)=s

      close 1
```

Program 20 *proxLoad.mps*

10.3.2 Term Cohesion

While statistical techniques may take into account term proximity as well as co-occurrence such as seen above, Salton [Salton 1983] suggests the following simpler formula for construction of term phrases (using global array notation corresponding to the arrays in the toolkit):

```
^Cohesion(i,j) = SIZE_FACTOR * (^tt(i,j)/(^dict(i)*^dict(j)))
```

That is, the sum or the co-occurrences between term *i* and term *j* divided by the product of the total number of occurrences of term *i* and term *j.*

The code to perform the cohesion calculation is shown in Program 21. In the script file *index.script,* it is invoked with:

cohesion.mps < term-term-matrix.sorted.txt | \

sort -nr > term-term-cohesion-similarities.txt

where the input, *term-term-matrix.sorted.txt,* is the output of the term-term correlation program *tt1.mps*.

```
#!/usr/bin/mumps
# ~/Medline2012/cohesion.mps
# Copyright 2016, 2022 Kevin C. O'Kane

# Feb 12, 2022

     zmain

# phrase construction

 for  do
 . read a
 . if '$test break
 . set w1=$piece(a," ",2)
 . set w2=$piece(a," ",3)
 . set co=$piece(a," ",4)
 . set ch=co/(^dict(w1)*^dict(w2))*100000\1
 . if ch>0 write ch," ",w1," ",w2,!
```

Program 21 *chohesion.mps*

Figure 60 gives a sample of the results.

```
1730 compute ct
1792 blot gene
1851 inflammate inflammatory
1881 gene messenger
1893 ne norepinephrine
1893 painful should
1923 endothelial endothelium
1949 mammography screene
2097 lymph node
2142 imag resonance
2307 imag magnetic
2747 echocardiography two-dimension
2870 colit ulcerative
2976 electron microscopy
3137 compute tomography
3846 magnetic resonance
3921 america society
4545 hypoxic normoxic
4901 dismutase superoxide
7500 aureus staphylococcus
8928 expiratory forc
14285 dodecyl electrophoresis
```

Figure 60 Term Term Cohesion Results

However the user should be cautioned that this procedure can sometimes result in unwanted connections such as *Venetian blind* and *blind Venetian*. For that reason, the aggregate relative position of the terms, as shown in the proximity calculation above, are useful in deciding when two terms are consistently linked. That is, if the order is strongly in favor of one term preceding another, this indicates a probable phrase; on the other hand, if the relative order is in neither direction, this is probably not a phrase.

10.4 Term-Term Matrix

10.4.1 Term-Term-Term Correlations

Term-Term-Term Connection Matrix

A term-term connection matrix gives second order term connections. That is, if some term *A* is related to some term *B* and term *B* is related to term *C*, there may be a relationship between terms *A* and *C* if the connections are sufficiently strong. It is calculated as follows:

$$tt_{km}^{2}=\sum_{p=1}^{N}\sum_{i=1}^{N}\sum_{j=1}^{N}d_{ik}d_{pk}d_{pm}d_{jm}$$

where *p, i,* and *j* are document numbers and *k* and *m* are terms. Again, as above, the values of d_{ik} are zero or one indicating if term *k* is present in document *i*.

For any matrix element in tt^2 addressed by row term *k* and any column term *m*, the value of the is the number of intermediate terms they have in common. Thus, for example, if some term *A* does not co-occur with term *B* but they both co-occur with one term *C*, the value of the element would be 1 and there is a weak indirect connection between term *A* and term *C*. Higher numbers imply stronger connections.

Program 22 shows calculation of a term-term connection matrix. A sample of the output is given in Figure 61.

```
#!/usr/bin/mumps

    zmain

# Copyright 2014 Kevin C. O'Kane

# ttt.mps Feb 12, 2022

 kill ^ttt

 for w1=$order(^tt(w1)) do
 . for w2=$order(^tt(w1,w2)) do
 .. for w3=$order(^tt(w2,w3)) do
 ... if w1=w3 break
 ... if '$data(^ttt(w1,w3)) set ^ttt(w1,w3)=1
 ... else set ^ttt(w1,w3)=^ttt(w1,w3)+1

 open 1:"term-term-term.txt,new"
 use 1
 for w1=$order(^ttt(w1)) do
 . for w2=$order(^ttt(w1,w2)) do
 .. write ^ttt(w1,w2)," ",w1," ",w2,!
 close 1
```

Program 22 *ttt.mps*

```
1 dystrophy deletion
1 extracorporeal calculi
1 fertilization cleave
1 fertilize fertilization
1 iron hydroxyl
1 oocyte cleave
1 oocyte fertilization
1 oocyte fertilize
1 transluminal ptca
```

Figure 61 Term-term-term Correlations

10.4.2 Term Clusters

Related terms can be grouped into clusters using the results of the term-term matrix. The program to do this is shown in Program 23. This program implements a simple single link clustering algorithm. That is, a term is added to a cluster if it is related to at least one term already in the cluster.

The input to this program is the term-term relationships contained in *^tt()* calculated in Program 18 above. The other term-term relationships may be tried as well although the proximity based relations give poor results.

The program works as follows. A temporary file is created whose lines consist of the a term-term correlation score followed by the two correlated words, blank separated. Words whose correlation scores are below a threshold are not written. The file is sorted according to the correlation score in reverse order (the highest score is at the beginning of the file, the lowest at the end).

The file is read line by line. Initially, the highest scoring word pair are placed in a cluster of their own.

Next successive lines are read. The words on each line compared to the words in the existing clusters. If one of the two words is in a cluster, the other word is added to the cluster. If neither word is in an existing cluster, a new cluster is created containing the two words.

At the end, the clusters are printed.

```
#!/usr/bin/mumps
# clustertt.mps
# Copyright 2014 Kevin C. O'Kane

# Feb 15, 2022

             zmain

 kill ^clstr
 kill ^x

 if '$data(%1) set min=.1
 else set min=%1

 open 1:"TTtmp.tmp,new"
 use 1
 for w1=$order(^tt(w1)) do
 . for w2=w1:$order(^tt(w1,w2)):"" do
 .. if ^tt(w1,w2)<min quit
 .. write ^tt(w1,w2)," ",w1," ",w2,!
 close 1

 shell sort -n -r < TTtmp.tmp > TTtmp.sorted.tmp

 open 1:"TTtmp.sorted.tmp,old"

 set c=1
 for  do
 . use 1
 . read a  // correlation word1 word2
 . if '$test break
 . set score=$p(a," ",1)
 . set w1=$p(a," ",2)
 . set w2=$p(a," ",3)
 . if w1=w2 quit
 . set f=1

# ^x() is a two dimensional array that contains, at the second level,
# a list of clusters to which the word (w1) belongs
# ^cluster() is the cluster matrix.  Each row (s) is a cluster
# numbered 1,2,3 ... The second level is a list of the words
# in the cluster.

# The following
# code runs thru all the clusters first for w1 (w1) and
# adds w2 to those clusters w1 belongs to.  It
# repeats the process for w2.  If a word pair are not
# assigned to some cluster (f=1), they are assigned to a new
# cluster and the cluster number is incremented (c)

 . if $d(^x(w1)) for s=$order(^x(w1,s)) do
 .. set ^clstr(s,w2)=""
 .. set ^x(w2,s)=""
 .. set f=0

 . if $d(^x(w2)) for s=$order(^x(w2,s)) do
 .. set ^clstr(s,w1)=""
 .. set ^x(w1,s)=""
 .. set f=0

 . if f do
 .. set ^clstr(c,w1)="" set ^x(w1,c)=""
 .. set ^clstr(c,w2)="" set ^x(w2,c)=""
 .. set c=c+1

# print the clusters

 close 1
 use 5
 write "number of clusters: ",c,!!
 for cx=$order(^clstr(cx)) do
 . write "cluster: ",cx,!
 . for w1=$order(^clstr(cx,w1)) do
 .. use 5 write w1," "
 . write !!

 halt

 kill ^dt
 kill ^dtt

 for d=$order(^doc(d)) do
 . for w=$order(^doc(d,w)) do
 .. if $data(^x(w)) do
 ... for c=$order(^x(w,c)) do
 .... if $data(^dt(d,c)) set ^dt(d,c)=^dt(d,c)+1
 .... else  set ^dt(d,c)=1

 for d=$order(^dt(d)) do
 . for c=$order(^dt(d,c)) do
 .. for w=$order(^clstr(c,w)) do
 ... if $data(^dtt(d,w)) set ^dtt(d,w)=^dtt(d,w)+1
 ... else  set ^dtt(d,w)=1

 for d=$order(^dtt(d)) do
 . write ^title(d),!
 . for w=$order(^dtt(d,w)) write w," "
 . write !
```

Program 23 Term-Term clustering

Figure 62 gives a sample of the output of the above. In this example, there 438 clusters were found.

```
164 cluster
adenocarcinoma adenoma colonic colorectal crohn hypercalcemia hyperparathyroidism metastasis parathyroid
parathyroidectomy pth

165 cluster
anemia aplastic ferritin iron overload sickle transferrin

166 cluster
extension femore flexion insertion invasion motion rotation

167 cluster
angina anginal angiographic anomaly arrest arrhythmia atrioventricular atrium axis cardiomyopathy
catheterization cava congestive echocardiographic echocardiography ejection glob hypertrophy
radionuclide sept shortene ventriculography

168 cluster
abscess arteriovenous clos closure cosmetic fistula flap incision suture

169 cluster
ace acetylcholine ach adenosine adherence adrenergic angina anginal angiographic angioplasty angiotensin
antagonize arrest arrhythmia atherosclerotic atrioventricular attenute autonomic balloon beat blocker
canine capill cardiomyopathy circumflex compete congestive conscious contractile contractility
contraction creatine cyclase descend dihydropyridine diuretic dog ejection endothelial glob glomerular
guinea inflation inotropic intra isometric junction lad mongrel myocardium narrow nephron norepinephrine
occlude ouabain patent perfus phentolamine pig postischemic potency pressor propranolol pump
radionuclide relaxation reperfus reperfusion restenosis resuscitte revascularization shortene stenos
strip sympathetic transluminal treadmill vasoconstrictor ventriculography wire yohimbine

17 cluster
acting activate activator adenosine adenylate adrenergic amp autoimmune band blot camp carboxyl cdna
complementary construct cyclase cyclic deduce deletion dodecyl domain dot enhancer epitope exon family
fibroblast fibronectin forskolin glycoprotein granulocyte gt11 homolog homology hybridization killer
kilobase lambda lectin libr mammalian mononuclear monophosphate mutant northern nucleotide
oligonucleotide perfus phosphodiesterase polypeptide precursor residue restriction screen sera
substitution sulfate synthesize terminu tran transcription transmembrane vector western

170 cluster
allele allogeneic basement bear biologic blot carrier cd3 cd4 cd8 comple haplotype helper
histocompatible hybridize immunofluorescence l3t4 lambda leu linkage lyt mitogen mitogenic monocyte
phenotype phytohemagglutinin polypeptide rearrange restriction southern spleen subset

171 cluster
avidin biotin glutathione paraffin peroxidase
```

```
172 cluster
arteriosus ductus occlude patent

173 cluster
actuarial adjuvant cisplatin cours distant irradiate radiotherapy squam

174 cluster
alzheimer behaviore chorionic cognitive cortex dementia gestational her home intrauterine labor memory
ment mother neuropsychological perinatal pregnant retardate tangle task trimester

175 cluster
ace acetylcholine ach adherence angiotensin antagonize attenute blocker compete contractile
contractility contraction dihydropyridine diuretic endothelial endothelium guinea inotropic isometric
junction norepinephrine pig potency propranolol relax relaxation strip yohimbine
```

Figure 62 Term Clusters

10.4.3 Adding Clusters to the Collection

It is possible (but not done in the toolkit) to replace original terms in the documents that occur in one or more clusters with a token indicating the cluster(s) to which they belong and then using these tokens as though they were terms. This approach can increase recall considerably but at the expense to precision. Alternatively, words from clusters can be added to search vectors.

10.4.4 Jaccard Term-Term Similarity

A similar measure can be calculated using the Jaccard metric. The formula (using global array notation) is:

```
set ^jaccardtt(i,j)=^tt(i,j)/(^df(i)+^df(j)-^tt(i,j))
```

That is, the Jaccard similarity between two words *i* and *j* is equal to the sum of the co-occurrences between term *i* and term *j* divided by the sum of the number of documents term *i* and term *j* occur in minus the sum of the co-occurrences of the terms.

Program 24 gives an example (not, the jaccard similarities are loaded in a separate program, *jLoad.mps* as shown. A sample of the output (using word stems) is shown in Figure 58.

```
#!/usr/bin/mumps
# jaccard-tt.mps
# Copyright 2014, 2022 Kevin C. O'Kane

# Feb 23, 2022

       zmain

      if '$data(%1) set JMIN=0
      else  set JMIN=%1

# input format: wgt word1 word2 co-occur-ww-w2 ^df(w1) ^df(w2)
# output format: jaccardtt word1 word2

       open 1:"term-term-jaccard-similarities.tmp,new"

       for  do
       . use 5 read a
       . if '$test break
       . set w1=$piece(a," ",2)
       . set w2=$piece(a," ",3)
       . set co=$piece(a," ",4)
       . set d1=$piece(a," ",5)
       . set d2=$piece(a," ",6)
       . set jc=co/(d1+d2-co)
       . if jc<JMIN quit
       . set jc=$j(jc,1,3)
       . use 1 write jc," ",w1," ",w2,!
       . set ^jtt(w1,w2)=jc
       . set ^jtt(w2,w1)=jc

       use 5
       close 1

-----
#!/usr/bin/mumps
# ~/Medline2012/jLoad.mps December 10, 2014
# Copyright 2014 Kevin C. O'Kane

       zmain

# input format: #co-occur w1 w2
```

```
      for  do
      . use 5
      . read a
      . if '$test break
      . set wgt=$piece(a," ",1)
      . set w1=$piece(a," ",2)
      . set w2=$piece(a," ",3)
      . set ^jtt(w1,w2)=wgt
```

Program 24 *jaccardtt.mps and jLoad.mps*

10.5 Cohesion Term Correlation

10.6 Adding Phrases to the Collection

Phrases may be added to the document-term matrix, the term-document matrix and to augment search queries. Several methods of detecting term-term relationships were discussed above. Of these, we will use the phrases based on word proximity determined by *proximity.mps* shown in Program 19 shown on page 140 rather than to other methods discussed above. Phrases are added to the collection in *proxLoad.mps* as shown in Program 20.

Program 20 creates two new global array matrices ^*ttp()* and ^*phrase()*. It also adds phrase entry terms to ^*doc(),* ^*index(),* ^*df()* and ^*dict().*

11 Document-Document Matrix

11.1 Document Clusters

The program *clusterdd.mps* in Program 25 uses a single link clustering technique similar to that used in the term clustering discussed above (Section 10.4.2 on page 144).

The program first generates and then reads a file of document-document correlations sorted in reverse (highest to lowest) correlation order. There is a filter on line 19 which ignores document-document correlation cosines that are below a given threshold (*min*, a command line parameter).

```
#!/usr/bin/mumps
# clusterdd.mps
# Copyright 2014, 2022 Kevin C. O'Kane

# Feb 25, 2022

     zmain

 kill ^clstr
 kill ^clv
 kill ^x
 kill ^ct
 kill ^dc

 if '$data(%1) write "Missing parameter",! halt

 set min=%1
 if min="" set min=0.3

 open 1:"ddtmp.tmp,new"
 use 1
 for d1=$order(^dd(d1)) do
 . for d2=d1:$order(^dd(d1,d2)):"" do
 .. if ^dd(d1,d2)<min quit
 .. write ^dd(d1,d2)," ",d1," ",d2,!
 close 1
 shell sort -n -r < ddtmp.tmp > ddtmp.sorted.tmp
 open 1:"ddtmp.sorted.tmp,old"
 set c=1
 for  do
 . use 1
 . read a  // correlation doc1 doc2
 . if '$test break
 . set score=$p(a," ",1)
 . set seq1=$p(a," ",2)
 . set seq2=$p(a," ",3)
 . if seq1=seq2 quit
 . set f=1

# ^x() is a two dimensional array that contains, at the second level,
# a list of clusters to which the document number (seq1) belongs
# ^cluster() is the cluster matrix.  Each row (s) is a cluster
# numbered 1,2,3 ... The second level is a list of the document
# numbers of those documents in the cluster. The following
# code runs thru all the clusters first for doc1 (seq1) and
# adds seq2 (doc2) to those clusters doc1 belongs to.  It
# repeats the process for seq2 (doc2).  If a doc pair are not
# assigned to some cluster (f=1), they are assigned to a new
# cluster and the cluster number is incremented (c)

 . if $d(^x(seq1)) for s="":$order(^x(seq1,s)):"" do
 .. set ^clstr(s,seq2)=""
 .. set ^x(seq2,s)=""
 .. set f=0

 . if $d(^x(seq2)) for s="":$order(^x(seq2,s)):"" do
 .. set ^clstr(s,seq1)=""
 .. set ^x(seq1,s)=""
 .. set f=0

 . if f do
 .. set ^clstr(c,seq1)="" set ^x(seq1,c)=""
 .. set ^clstr(c,seq2)="" set ^x(seq2,c)=""
 .. set c=c+1

# print the clusters

 close 1
 use 5
 open 1:"document-clusters.txt,new"
 open 2:"document-clusters-table.txt,new"

 for cx=$order(^clstr(cx)) do
 . set mx=0
 . for d1=$order(^clstr(cx,d1)) do   // doc in cluster
 .. set x=0
 .. for w=$o(^doc(d1,w)) do
 ... set x=x+^idf(w)
 .. if x>mx set mx=x,dx=d1
 .. set ^ct(cx,d1)=""
 .. if $data(^dc(d1,cx)) set ^dc(d1,cx)=^dc(d1,cx)+1
 .. else  set ^dc(d1,cx)=1
 . u 1 w cx," ",mx," "
 . u 2 w "cluster: ",cx,!
 . u 1 write $e(^title(dx),1,70),!!
 . set ^clstr(cx)=^title(dx)
 . open 3:"tmp,new"
 . for w=$o(^doc(dx,w)) do
 .. use 1 write w,"(",^doc(dx,w),") " if $x>60 w !
 .. use 3 write ^doc(dx,w)," ",w,!
 . u 1 write !!
 . u 3 write ! close 3
 . shell sort -nr < tmp > tmp1
 . open 3:"tmp1,old"
 . for  do
 .. u 3 read t if '$t break
 .. u 1 w t," "
 . c 3
 . u 1 w !!
  . for d1=$order(^clstr(cx,d1)) do
  .. u 1 write d1," ",$e(^title(d1),1,70),!
  .. u 2 write d1," ",$e(^title(d1),1,70),!
  . u 1 w "----------------",!!
  . u 2 w !

  close 1
  close 2

  open 1:"document-clusters-by-title.txt,new"
  use 1
  for d=$order(^dc(d)) do
  . write d," ",^title(d),!,?10
  . for c=$order(^dc(d,c)) do
  .. write c," "
  . write !

  close 1
  use 5

```

Program 25 Document clustering

Figure 63 gives a sample output from Program 25.

```
25 137.39 The functional border zone in conscious dogs.

anesthetize(12.062) border(56.427) bound(25.748) circumflex(16.122)
conscious(13.292) coron(13.938) curve(10.44) distance(13.782)
dogs(20.948) dysfunction(16.806) extent(10.246) latere(11.836)
nonischemic(26.517) occlusion(21.904) open-chest(16.374) perfusion(22.068)
sigmoid(18.648) systolic(10.992) thickene(24.561) wall(15.711)
zone(50.778)

56.427 border 50.778 zone 26.517 nonischemic 25.748 bound 24.561 thickene 22.068 perfusion 21.904
occlusion 20.948 dogs 18.648 sigmoid 16.806 dysfunction 16.374 open-chest 16.122 circumflex 15.711 wall
13.938 coron 13.782 distance 13.292 conscious 12.062 anesthetize 11.836 latere 10.992 systolic 10.44
curve 10.246 extent

1264 Effect of lidocaine on the myocardial acidosis induced by coronary art
2174 Possible role of oxygen-derived, free radicals in cardiocirculatory sh
2288 Dissociation between global and regional systolic and diastolic ventri
2289 Time course of recovery of "stunned" myocardium following variable per
254 Hierarchy of levels of ischemia-induced impairment in regional left ve
266 Salutary action of nicorandil, a new antianginal drug, on myocardial m
267 The functional border zone in conscious dogs.
268 The physiologic basis of dobutamine as compared with dipyridamole stre
2715 Effects of isoflurane on myocardial blood flow, function, and oxygen c
3034 Controlled reperfusion following regional ischemia.
3505 Reductions in regional myocardial function at rest in conscious dogs w
3513 Effects of left ventricular receptor stimulation on coronary blood flo
3525 Reduction of reperfusion injury in the canine preparation by intracoro
3548 The rapid evolution of a myocardial infarction in an end-artery corona
3953 Dissociation between regional myocardial dysfunction and subendocardia
3954 End-systolic radius to thickness ratio: an echocardiographic index of
4133 Quantification of left-ventricular regional dyssynergy by radionuclide
947 Altered response of reperfused myocardium to repeated coronary occlusi
-----------------
```

Figure 63 Example from *document-clusters.txt*

```
cluster: 1
1340 Percutaneous renal calculus removal in an extracorporeal shock wave li
1343 Management of upper ureteral calculi with extracorporeal shock wave li
1370 Initial experience with extracorporeal shock wave lithotripsy in child
1401 Extracorporeal shock wave lithotripsy in childhood.

cluster: 10
2415 Dietary calcium intake and bone loss from the spine in healthy postmen
877 Dietary calcium intake and rates of bone loss in women.

cluster: 11
3257 Outcome of in-vitro fertilization and embryo transfer after different
3728 A correlative study on the embryotrophic property of patient's serum a
536 The effect of polyploidy on embryo cleavage after in vitro fertilizati
537 Fertilization and cleavage rates of heparin-exposed human oocytes in v

cluster: 12
2045 Percutaneous transfemoral placement of the Kimray-Greenfield vena cava
2046 Percutaneous insertion of the Kimray-Greenfield filter: technical cons

cluster: 13
1807 Preliminary report: effects of high dose methylprednisolone on delayed
1831 Predicting cerebral ischemia after aneurysmal subarachnoid hemorrhage:

cluster: 14
4192 Pharmacokinetics of systemically administered cocaine and locomotor st
4209 Cocaine disposition in the brain after continuous or intermittent trea

cluster: 15
3168 Cartilage proteoglycans in synovial fluid and serum in patients with i
3310 Diagnosis of pseudogout and septic arthritis.
4190 Influence of concomitant aspirin or prednisone on methotrexate synovia
```

Figure 64 Example from *document-clusters-table.txt*

```
1084 Syringomyelia: cyst measurement by magnetic resonance imaging and comparison with symptoms, signs
and disability.
          47
1230 Five years' experience with continuous ambulatory or continuous cycling peritoneal dialysis in
children.
          4
1253 Comparative analysis of beta-1 adrenoceptor agonist and antagonist potency and selectivity of
cicloprolol, xamoterol and pindolol.
          82
1259 Pharmacological analysis of the cardiac actions of xamoterol, a beta adrenoceptor antagonist with
partial agonistic activity, in guinea pig heart: evidence for involvement of adenylate cyclase s
          82
1264 Effect of lidocaine on the myocardial acidosis induced by coronary artery occlusion in dogs.
          25 44 46
1274 Regulation of alpha-1 adrenergic receptor density and functional responsiveness in rat brain.
          58
1276 Characterization of the vasoactive intestinal peptide receptor in rat submandibular gland:
radioligand binding assay in membrane preparations.
          57 58
1283 Characterization of D-2 dopamine receptors in a tumor of the rat anterior pituitary gland.
          57 58
```

Figure 65 Example from *document-clusters-table.txt*

Document clusters can be used during retrieval to display documents that might be related to documents retrieved as a result of a user query.

11.2 Document Hyper Clusters

A program to generate clusters of clusters is given in Program 26. This program can be time consuming on a large collection. This program builds, for each cluster, a centroid vector and then clusters the centroid vectors.

```
#!/usr/bin/mumps
# hyperCluster.mps
# Copyright 2014, 2022 Kevin C. O'Kane

# Feb 25, 2022

            zmain

 kill ^hc
 set c=0,k=0

 if '$data(%1) write "Missing parameter",! halt
 if '$data(%2) write "Missing parameter",! halt

 set min=%1
 set wgt=%2

 if min="" set min=0.4
 if wgt="" set wgt=.4

#     read the level one clusters and build
#     centroid vectors

 for  do
 . read a
 . if '$test break
 . if a="" quit
 . set t=$p(a," ",1)
 . if t="cluster:" do  quit
 .. for w=$order(^hc(c,w)) set ^hc(c,w)=^hc(c,w)/k
 .. set c=c+1,k=0
 .. quit
 . for w=$order(^doc(t,w)) do
 .. if $data(^hc(c,w)) set ^hc(c,w)=^hc(c,w)+^doc(t,w)
 .. else  set ^hc(c,w)=^doc(t,w)
 .. set k=k+1

 for i=1:1:c for w=$order(^hc(i,w)) if ^hc(i,w)<wgt kill ^hc(i,w)

#     write centroid vectors

 write !,"Centroid vectors",!!
 for i=1:1:c do
 . write i," "
 . for w=$order(^hc(i,w)) write w," (",$j(^hc(i,w),3,2),") "
 . write !!

 open 1:"hypertmp.tmp,new"

#     calculate cluster similarities

 write !!,"Cluster similarities:",!
 for i=1:1:c do
 . for j=i+1:1:c do
 .. s x=$zzCosine(^hc(i),^hc(j))
 .. if x<min quit
 .. use 5 write i," ",j," ",x,!
 .. use 1 write x," ",i," ",j,!

 use 5
 close 1

 kill ^clstr
 kill ^x

 shell sort -n -r < hypertmp.tmp > hypertmp.sorted.tmp

 open 1:"hypertmp.sorted.tmp,old"
 set c=1
 for  do
 . use 1
 . read a  // correlation doc1 doc2
 . if '$test break
 . set score=$p(a," ",1)
 . set seq1=$p(a," ",2)
 . set seq2=$p(a," ",3)
 . if seq1=seq2 quit
 . set f=1

# ^x() is a two dimensional array that contains, at the second level,
# a list of clusters to which the document number (seq1) belongs
# ^cluster() is the cluster matrix.  Each row (s) is a cluster
# numbered 1,2,3 ... The second level is a list of the document
# numbers of those documents in the cluster. The following
# code runs thru all the clusters first for doc1 (seq1) and
# adds seq2 (doc2) to those clusters doc1 belongs to.  It
# repeats the process for seq2 (doc2).  If a doc pair are not
# assigned to some cluster (f=1), they are assigned to a new
# cluster and the cluster number is incremented (c)

 . if $d(^x(seq1)) for s=$order(^x(seq1,s)) do
 .. set ^clstr(s,seq2)=""
 .. set ^x(seq2,s)=""
 .. set f=0

 . if $d(^x(seq2)) for s=$order(^x(seq2,s)) do
 .. set ^clstr(s,seq1)=""
 .. set ^x(seq1,s)=""
 .. set f=0

 . if f do
 .. set ^clstr(c,seq1)="" set ^x(seq1,c)=""
 .. set ^clstr(c,seq2)="" set ^x(seq2,c)=""
 .. set c=c+1

# print the clusters

 close 1
 use 5
 kill ^h


 for cx=$order(^clstr(cx)) do
 . open 1:"cterms.tmp,new"
 . for w=$order(^hc(cx,w)) do
 .. u 1 w ^hc(cx,w)," ",w,!
 . close 1
 . shell sort -nr <cterms.tmp > ctermssorted.tmp
 . set a=""
 . open 1:"ctermssorted.tmp,old"
 . u 1 for i=1:1:9 r x q:'$t  set a=a_" "_$o(^stem($p(x," ",2),""))
 . set ^clstr(cx)=a
 . close 1

 u 5

 write !!,"Number of clusters: ",c-1,!!
 for cx=$order(^clstr(cx)) do
 . write "cluster: ",cx,!
 . set ^h(cx)=^clstr(cx)
 . for seq1=$order(^clstr(cx,seq1)) do
 .. write "base cluster=",seq1,!
 .. set ^h(cx,seq1)="Cluster "_seq1
 .. for cz=$order(^ct(seq1,cz)) do
 ... write seq1,?8,$e(^title(cz),1,70),!
 ... set ^h(cx,seq1,cz)=""
 . write !
```

Program 26 Document hyper-clusters

```
Number of clusters: 7

cluster: 1

base cluster=14
14      Pharmacokinetics of systemically administered cocaine and locomotor st
14      Cocaine disposition in the brain after continuous or intermittent trea

base cluster=21
21      Tricuspid regurgitation in patients with acquired, chronic, pure mitra
21      Clinical experience with the Bjork-Shiley integral monostrut heart val
21      Simultaneous implantation of St. Jude Medical aortic and mitral prosth
21      Repair of tricuspid valve insufficiency in patients undergoing double
21      The choice of anticoagulation in pediatric patients with the St. Jude

base cluster=33
33      Cardiovascular risk factors from birth to 7 years of age: the Bogalusa
33      Cardiovascular risk factors from birth to 7 years of age: the Bogalusa

base cluster=40
40      Effect of beta 1-receptor blockade on coronary resistance in partially
40      Time course of recovery of "stunned" myocardium following variable per
40      The functional border zone in conscious dogs.
40      The physiologic basis of dobutamine as compared with dipyridamole stre
40      Effects of isoflurane on myocardial blood flow, function, and oxygen c
40      Reductions in regional myocardial function at rest in conscious dogs w
40      Effects of left ventricular receptor stimulation on coronary blood flo
40      Dissociation between regional myocardial dysfunction and subendocardia
```

Program 27 Example Document Hyper-Clusters

12 SQL Database Alternative

12.1 Relational Database Implementation

In a relational implementation, mapping the document-term matrix to a two dimensional table such as that shown in Figure 17 where the rows represent documents and the columns vocabulary terms and where NULLs are stored where a given row/column cell does not exist. This would result in an impractically large and inefficient table.

On the other hand, if the document-term matrix is mapped to a three column table where one column is the document number, one column is an indexing word[25] occurring in the document, and the final column is the frequency of occurrence[26] of the indexing word in the document, the problem becomes more manageable.

While the number of rows in the table would be quite large, most RDBMS systems are capable of efficiently indexing tables.

In the implementation discussed below, one of the output files consists of SQL commands to create a relational database for the OHSU test collection. A sample is shown in Figure 66.

25 Or a numeric stand-in token for same.

26 Or a value indicating the importance of the term in the document.

```
drop table if exists docVect;
create table docVect (doc int, word varchar(100), wgt float);
drop table if exists docs;
create table docs (doc int, title varchar(255));
insert into docs values (1, 'The binding of acetaldehyde to the active site of ribonuclease:
alterations in catalytic activity and effects of phosphate.');
insert into docVect values (1, 'absence', 23.885);
insert into docVect values (1, 'adduct', 17.654);
insert into docVect values (1, 'amino', 27.964);
insert into docVect values (1, 'aris', 15.094);
insert into docVect values (1, 'bind', 13.872);
insert into docVect values (1, 'catalytic', 17.27);
insert into docVect values (1, 'diminish', 12.698);
insert into docVect values (1, 'formation', 10.1);
insert into docVect values (1, 'peak', 10.16);
insert into docVect values (1, 'phosphate', 68.27);
insert into docVect values (1, 'protect', 11.316);
insert into docVect values (1, 'residue', 30.188);
```

Figure 66 SQL Database Creation Commands

Access to and manipulation of records in this database is by means of SQL queries. Some queries, as seen in Chapter Error: Reference source not found, can be quite complex.

Figure 67 shows an example. The contents of the SQL file (named *SQL.commands*) produced by the indexing scheme discussed below, a fragment of which is shown in Figure 66, were loaded into Sqlite3 and the query:

```
select * from docVect where word='diagnose';
```

was entered. The results are in Figure 67. The first field is the document number, the second field is the word and the third field is the weight of the word in the document.

Other SQL commands may be used to join, project, and so forth.

```
100|diagnose|10.13
1350|diagnose|10.13
162|diagnose|15.195
218|diagnose|10.13
2203|diagnose|20.26
2489|diagnose|10.13
2687|diagnose|10.13
310|diagnose|10.13
3253|diagnose|10.13
3453|diagnose|10.13
3597|diagnose|50.65
3648|diagnose|15.195
3921|diagnose|10.13
3949|diagnose|10.13
4714|diagnose|10.13
556|diagnose|10.13
```

Figure 67 Sqlite3 Query Output

12.2 Relational Database Creation

The program *SQLdocvectors.mps* (see Program 28) generates SQL commands to build a relational database examples of which are shown in Figure 68. Program 28 contains some examples as comments at the end. The commands are quitable for use with SQLite3.

```
#!/usr/bin/mumps
# SQLdocvectors.mps January 13, 2016
# Copyright 2016 Kevin C. O'Kane

       zmain

 write "use isr;",!
 write "begin transaction;",!
 write "drop table if exists docVect;",!
 write "create table docVect (doc int, word varchar(100), wgt float);",!
 write "drop table if exists docs;",!
 write "create table docs (doc int, title varchar(255));",!
 for d=$order(^doc(d)) do
 . if $order(^doc(d,""))="" quit
 . set t=$translate($extract(^title(d),1,256),"'")
 . use 5 write "insert into docs values (",d,", '",t,"');",!
 . for w=$order(^doc(d,w)) do
 .. write "insert into docVect values (",d,", '",w,"', ",^doc(d,w),");",!
 write "drop table if exists terms;",!
 write "create table terms (word varchar(100), TotCount int, DocCount int);",!
 for w=$order(^dict(w)) do
 . write "insert into terms values ('",w,"',",^dict(w),",",^df(w),");",!

 write "drop table if exists dd;",!
 write "create table dd (doc1 int, doc2 int, wgt float);",!

 for d1=$order(^dd(d1)) do
 . for d2=$order(^dd(d1,d2)) do
 .. write "insert into dd values (",d1,",",d2,",",^dd(d1,d2),");",!

 write "drop table if exists tt;",!
 write "create table tt (term1 varchar(100), term2 varchar(100), wgt float);",!

 for t1=$order(^tt(t1)) do
 . for t2=$order(^tt(t1,t2)) do
 .. write "insert into tt values ('",t1,"','",t2,"',",^tt(t1,t2),");",!

 write "commit transaction;",!

# examples
# create table docs (doc int, title varchar(255))
# create table docVect (doc int, word varchar(100), wgt float);
# create table terms (word varchar(100), TotCount int, DocCount);
# create table dd (doc1 int, doc2 int, wgt float);
# create table tt (term1 varchar(100), term2 varchar(100), wgt float);

# select doc,word, wgt from docVect
#      where word='mole' or word='ethylate'
#      order by wgt desc;

#  doc |   word   |  wgt
# -----+----------+-------
#    1 | ethylate | 37.95
#  242 | mole     |   6.9
#  857 | mole     |   6.9
# (3 rows)


# select a.doc,a.wgt, docs.title from docs natural join
#      (select doc, round(sum(wgt)) as wgt from docVect
#              where word='mole' or word='ethylate'
#              group by doc order by wgt desc) as a;
#
#  doc | wgt |                                             title
# -----+-----+---------------------------------------------------------------------------------------------
#    1 |  38 | The binding of acetaldehyde to the active site of ribonuclease  alteration
#  242 |   7 | Planar bilayer reconstitution of calcium channels  lipid effects on single
#  857 |   7 | The O-linked oligosaccharide structures are striking different on pregnancy
# (3 rows)

# select d1.doc, d2.doc, count(d1) as count from docVect as d1, docVect as d2
# where d1.word=d2.word and d1.doc != d2.doc group by d1.doc,d2.doc order by count desc;

# select doc, round(sum(wgt)) as wgt from docVect
# where word='mole' or word='ethylate' group by doc order by wgt desc;
```

Program 28 *SQLdocvectors.mps*

```
drop table if exists docVect;
create table docVect (doc int, word varchar(100), wgt float);
drop table if exists docs;
create table docs (doc int, title varchar(255));
insert into docs values (1, 'The binding of acetaldehyde to the active site of ribonuclease: alterations in catalytic
activity and effects of phosphate.');
insert into docVect values (1, 'absence', 23.885);
insert into docVect values (1, 'adduct', 17.654);
insert into docVect values (1, 'amino', 27.964);
insert into docVect values (1, 'aris', 15.094);
insert into docVect values (1, 'bind', 13.872);
insert into docVect values (1, 'catalytic', 17.27);
insert into docVect values (1, 'diminish', 12.698);
insert into docVect values (1, 'formation', 10.1);
insert into docVect values (1, 'peak', 10.16);
insert into docVect values (1, 'phosphate', 68.27);
insert into docVect values (1, 'protect', 11.316);
insert into docVect values (1, 'residue', 30.188);
insert into docs values (100, 'Edrophonium provocation test in the diagnosis of diffuse oesophageal spasm.');
insert into docVect values (100, 'baseline', 11.504);
insert into docVect values (100, 'diagnose', 10.13);
insert into docVect values (100, 'diffuse', 12.93);
insert into docVect values (100, 'manometry', 49.878);
insert into docVect values (100, 'oesophageal', 24.939);
insert into docVect values (100, 'positive', 16.968);
insert into docVect values (100, 'provocate', 52.962);
insert into docVect values (100, 'typical', 13.094);
```

Figure 68 *SQLcommands*

13 Retrieval

13.1 Vector Space Based Retrieval

Having constructed the basic matrices and vectors, we now explore some simple retrieval methods.

13.1.1 Retrieval Using the Doc-Term Matrix

A very simple program to scan the document-term matrix looking for documents that have terms from a query vector is given in Program 29.

In this program, a line of query terms is read. The terms are converted to lower case and stemmed. Next, the documents are scanned sequentially and, if found to have any one of the query terms, the title and document number are printed[27]. If a document contains more than one query vector term, it will be displayed twice. This program may operate in *read-only* mode as it does not change the database.

27 Note: titles may not necessarily contain any of the search terms.

```
#!/usr/bin/mumpsRO
# query1.mps Copyright 2014 Kevin C. O'Kane

 write "Enter search terms:",!
 set i=$zzInput("w")-1
 for j=0:1:i do
 . set a=$zlower(w(j))
 . set a=$zstem(a)
 . set query(a)=""

 set t1=$zd1
 for i=$order(^doc(i)) do
 . for j=$order(query(j)) do
 .. if $data(^doc(i,j)) write i,?8," ",^title(i),! break

 write !,"Time elapsed: ",$zd1-t1,!
 halt

Enter search terms:
epithelial fibrosis
1228    Energy expenditure of patients with cystic fibrosis.
1345    The occurrence of vasculitis in perianeurysmal fibrosis.
1486    Cellular and non-cellular compositions of crescents in human glomerulonephritis.
1504    Epithelial ovarian tumor in a phenotypic male.
1560    Neonatal long lines for intravenous antibiotic therapy in cystic fibrosis [letter]
1871    Intraperitoneal cis-platinum as salvage therapy for refractory epithelial ovarian
1881    The effect of a collagen bandage lens on corneal wound healing: a preliminary
1882    Removal of lens epithelial cells by ultrasound in endocapsular cataract surgery.
1886    Oncocytic carcinoma of the plica semilunaris with orbital extension.
1904    Retinal pigment epithelial cells release inhibitors of neovascularization.
```

Program 29 Simple retrieval program

13.1.2 Retrieval Using the Term-Doc Matrix

Unfortunately, Program 29 is very slow because all documents must be inspected. Program 30, on the other hand, is a simple program that uses the term-document matrix to more quickly search for documents that contain one or more search terms.

In this program, only those documents that actually contain a query term are inspected[28]. When a document containing one of the search terms is found, its document number and title are printed to *stdout*. However, as with Program 29, if a document contains more than one query term, it's title will be printed twice.

28 Note: the order of the titles will normally be different than in Program 29.

However, this can be fixed by routing the output through *sort* and *uniq* which will eliminate duplicates as well as give a count, for each title, of the number of times the title appeared. This number is also the number of terms from the query vector found in each document. This can be done with a command such as:

```
query2.mps < input_words.txt | sort | uniq -c | sort -nr > results
```

Where *input_words.txt* contains the search terms (one per line). The output from *query2.mps* is sorted so that any duplicate document number / title combinations will be on consecutive lines. The number of times a document number / title combination line appears indicates the number of keywords found in the document.

The *uniq* program removes duplicate lines and the *-c* option causes the remaining line to be prefixed by a count of the number of times the line appeared in the input file.

The final sort sorts the results based on the first field on each line which is now the count of the number of times the title appeared. The *sort* is such that those titles with the highest counts appear first and those with lower counts appear last (the *-n* option causes a numeric sort and the *-r* option reverses the order so that results display with the titles with higher counts appear first).

This is a primitive form of weighted results. Those documents with more search terms in common with the query are shown first. Many other weighting schemes are possible.

```
#!/usr/bin/mumpsRO
# query2.mps Copyright 2014 Kevin C. O'Kane

 write "Enter search terms:",!
 set i=$zzInput("w")-1
 for j=0:1:i do
 . set a=$zlower(w(j))
 . set a=$zstem(a)
 . set query(a)=""

 set t1=$zd1

 for w=$order(query(w)) do
 . for d=$order(^index(w,d)) do
 .. write d,?8," ",^title(d),!

 write !,"Time elapsed: ",$zd1-t1,!

Enter search terms:
epithelial fibrosis
1486    Cellular and non-cellular compositions of crescents in human glomerulonephritis.
1504    Epithelial ovarian tumor in a phenotypic male.
1871    Intraperitoneal cis-platinum as salvage therapy for refractory epithelial ovarian
1881    The effect of a collagen bandage lens on corneal wound healing: a preliminary
1882    Removal of lens epithelial cells by ultrasound in endocapsular cataract surgery.
1886    Oncocytic carcinoma of the plica semilunaris with orbital extension.
1904    Retinal pigment epithelial cells release inhibitors of neovascularization.
1924    Ocular pathologic findings in neonatal adrenoleukodystrophy.
2019    Metastatic adeno or undifferentiated carcinoma from an unknown primary site--natural

Elapsed time: 0
```

Program 30 Term-Doc matrix search

13.1.3 Weighted Retrieval using the Term-Doc Matrix

Program 31 is similar to Program 30 but rather than print the title of any document which contains any search term, it creates a primitive score for each document to indicate how similar the document is to the query vector. It does this by summing the weights of those terms from each document vector which are in common with the terms in the query vector. It then prints the titles of the 10 documents with the highest scores.

This is another form of primitive search weighting. If a title has several terms in common with the query vector and if these are highly weighted terms, the title is deemed to be a better match with the query that if it had only a few low weighted terms in common.

Note that in Program 31 that the **$job** builtin variable returns the process id of the currently running program. This is unique number and it is used to name the temporary file that contains the unsorted results.

Program 31 creates a temporary global array vector (^*tmp*) whose indices are the document numbers of those documents that have one or more terms in common with the query vector. The value of an element of the ^*tmo()* vector is the sum of the weights of the query terms in common with the document.

For example, if document 10 has three terms in common with the query and the terms have weights 2, 3 and 5, respectively, the value in ^*tmp(10)* will be 10. The weights in this example are assumed to be IDF weights which were discussed above although other weighting schemes may be used.

Program 31, like Program 30, iterates through the query vector and, for each word in the query vector, finds all those documents that contain the term. The weights of the query vector term in the document are added to the ^*tmp()* vector for the document.

After the totals are calculated in ^*tmp*, the scores (from ^*tmp)* and the corresponding titles are written to the temporary file. This is then sorted by score and the top ten are printed.

```
#!/usr/bin/mumps
# query3.mps Copyright 2014 Kevin C. O'Kane

 kill ^tmp

 write "Enter search terms:",!
 set i=$zzInput("w")-1
 for j=0:1:i do
 . set a=$zlower(w(j))
 . set a=$zstem(a)
 . set query(a)=""

 if $order(query(""))="" halt

 set t1=$zd1

 for w=$order(query(w)) do
 . for i=$order(^index(w,i)) do
 .. if $data(^tmp(i)) set ^tmp(i)=^tmp(i)+^index(w,i)
 .. else  set ^tmp(i)=^index(w,i)

 set fn=$job_",new"
 open 1:fn
 use 1
 for i=$order(^tmp(i)) do
 . use 1 write $justify(^tmp(i),6,2)," ",^title(i),!
 close 1
 use 5
 set cmd="sort -n <"_$job_" | tail -10 ; rm "_$job
 shell &~cmd~

 write !,"Elapsed time: ",$zd1-t1,!
 halt

Enter search terms:
zinc
 23.64 Effect of maternal zinc supply on blood and tissue metallothionein I concentrations
 23.64 Observations of serum trace elements in chronic lymphocytic leukemia.
 23.64 Zinc absorption in humans from meals based on rye, barley, oatmeal, triticale and
 35.46 Cadmium-induced immunopathology is prevented by zinc administration in mice.
 41.37 Magnesium and zinc status during the menstrual cycle.
 59.10 The relationship between cadmium, zinc, and birth weight in pregnant women who
 76.83 Iron and zinc concentrations and 59Fe retention in developing fetuses of zinc-defic
 82.74 The effect of smoking on placental and fetal zinc status.
 82.74 Zinc deficiency increases the osmotic fragility of rat erythrocytes.
100.47 Dietary conditions influencing relative zinc availability from foods to the rat and
```

Program 31 Weighted Term-Doc matrix search

13.1.4 Simple Cosine Weighted Retrieval Part 1

The previous programs used relatively simple weighting schemes based on counts and weight sums. These can be useful but fail deal with issues such as query and document vector lengths, among others.

More sophisticated methods take into account the number of terms in the search vector, the number of terms in the document vector and the number of terms in the intersection of the query and document vector.

Program 32 uses a more sensitive similarity metric. It reads a set of query words into a query vector then calculates the *cosines*[29] between the query and each document vector and, for those documents where the cosine is greater than zero, it writes the cosine and title to a temporary file. When all documents have been processed, it sorts the results in the temporary file based on their cosine scores and prints the titles of the 10 documents with the highest scores. A shell script is used to sort the result. Output from the shell script is written directly to the user's console.

29 See Basic Vector Similarity Functions on page 32.

```
#!/usr/bin/mumps
# simpleRetrieval.mps Copyright 2014 Kevin C. O'Kane

 kill ^query

 write "Enter search terms:",!

 set i=$zzInput("w")-1
 for j=0:1:i do
 . set a=$zlower(w(j))
 . set a=$zstem(a)
 . set ^query(a)=1

 if $order(^query(""))="" halt

 write "Query is: "
 for w=$order(^query(w)) write w," "
 write !

 set time0=$zd1

 set fn=$job_",new"
 open 1:fn
 use 1

 set x=0
 for i=$order(^doc(i))  do       // calculate cosine between query and each doc
 . if i="" break
 . set c=$zzCosine(^doc(i),^query)

# If cosine is > zero, put it and the doc offset (^doc(i)) into an answer vector.
# Make the cosine a right justified string of length 5 with 3 digits to the
# right of the decimal point.  This will force numeric ordering on the first key.

 . if c>0 write $justify(c,5,3)," ",^title(i),! set x=x+1

 use 5
 close 1

 write x," documents found - showing top 10 results:",!!

 set cmd="sort -n <"_$job_" | tail -10 ; rm "_$job
 shell &~cmd~

 write !,"Time used: ",$zd1-time0," seconds",!

 halt
```

```
Enter search terms:
zinc
Query is: zinc
27 documents found - showing top 10 results:

0.438 Observations of serum trace elements in chronic lymphocytic leukemia.
0.482 Pulmonary involvement in zinc fume fever.
0.511 Magnesium and zinc status during the menstrual cycle.
0.515 Multiple sclerosis and the workplace: report of an industry-based cluster.
0.526 Iron and zinc concentrations and 59Fe retention in developing fetuses of zinc-deficie
0.565 Diagnostic value of zinc levels in pleural effusions [letter]
0.628 Zinc deficiency increases the osmotic fragility of rat erythrocytes.
0.641 The relationship between cadmium, zinc, and birth weight in pregnant women who smoke.
0.693 The effect of smoking on placental and fetal zinc status.
0.810 Dietary conditions influencing relative zinc availability from foods to the rat and

Time used: 3 seconds
```

Program 32 Simple cosine based retrieval

13.1.5 Simple Cosine Weighted Retrieval Part 2

As before, program 32 is slow because it calculates the cosines between all documents and the query vector. In fact, most documents contain no words in common with the query vector and, consequently, their cosines are zero. Thus, a possible speedup technique would be to only calculate the cosines between the query and those documents that contain at least one term in common with the query vector.

We do this by first constructing a vector containing document numbers of those documents with at least one term in common with the query. This is shown in Program 33.

For each query term *w*, the document numbers *d* from row *^index(w)* are used as indices to create elements of *^tmp(d)*. No value is stored at the nodes as the index (the document number) is the information in which we are interested. The *^tmp()* vector a list of document numbers of those documents containing at least one query term. If a document has several query terms, it still has only one element in*^tmp()*. The number of elements of *^tmp* will normally be considerably smaller than the total number of documents[30].

When all query words have been processed, the temporary vector *^tmp* indices are those document numbers of documents that contain at least one query term. The cosinses are then calculated between the query vector and only those documents whose document numbers are indices of *^tmp()*.

Program 33 yields the same results as above but takes less than 1 second.

30 Unless the query vector consists of a large list of terms that are widely distributed among the document collection. This would be a very non-specific query, to say the least!

```
#!/usr/bin/mumps
# fasterRetrieval.mps Copyright 2014 Kevin C. O'Kane

 kill ^query
 kill ^tmp

 set fn=$job_",new"
 open 1:fn

 write "Enter search terms:",!

 set i=$zzInput("w")-1
 for j=0:1:i do
 . set a=$zlower(w(j))
 . set a=$zstem(a)
 . set ^query(a)=1

 if $order(^query(""))="" halt

 write "Query is: "
 for w=$order(^query(w)) write w," "
 write !

 set time0=$zd1

# Find documents containing one or more query terms.

 for w=$order(^query(w)) do
 . for d=$order(^index(w,d)) set ^tmp(d)="" // retain doc id

 use 1
 set x=0

 for i=$order(^tmp(i)) do  // calculate cosine between query and each doc
 . set c=$zzCosine(^doc(i),^query)   // MDH cosine calculation

# If cosine is > zero, puAt it and the doc offset (^doc(i)) into an answer vector.
# Make the cosine a right justified string of length 5 with 3 digits to the
# right of the decimal point.  This will force numeric ordering on the first key.

 . if c>0 write $justify(c,5,3)," ",^title(i),! set x=x+1

 use 5
 close 1

 write x," documents found - showing top 10 results:",!!
 set cmd="sort -n <"_$job_" | tail -10 ; rm "_$job
 shell &~cmd~

 write !,"Time used: ",$zd1-time0," seconds",!
 halt

Enter search terms:
zinc
Query is: zinc
27 documents found - showing top 10 results:

0.438 Observations of serum trace elements in chronic lymphocytic leukemia.
0.482 Pulmonary involvement in zinc fume fever.
0.511 Magnesium and zinc status during the menstrual cycle.
0.515 Multiple sclerosis and the workplace: report of an industry-based cluster.
0.526 Iron and zinc concentrations and 59Fe retention in developing fetuses of zinc-defic
0.565 Diagnostic value of zinc levels in pleural effusions [letter]
0.628 Zinc deficiency increases the osmotic fragility of rat erythrocytes.
0.641 The relationship between cadmium, zinc, and birth weight in pregnant women who smoke.
0.693 The effect of smoking on placental and fetal zinc status.
0.810 Dietary conditions influencing relative zinc availability from foods to the rat and

Time used: 0 seconds
```

Program 33 Faster simple retrieval

13.1.6 Retrieval Enhancements

The code from Program 33 may be enhanced as shown in Program 34. This version, for each query term:

1. gives any other terms in the word stem group for the term,
2. augments the query with related terms, and
3. gives a list of terms that may sound similar to the query term.

The terms added to the query are based on term proximity. The list of sound-alike terms is based on Soundex codes.

```
#!/usr/bin/mumps
# ~/Medline2012/medlineRetrieve.mps Nov 12, 2014
# Copyright 2014 Kevin C. O'Kane

       kill ^query
       kill ^ans
       kill ^tmp

       for  do                    // extract query words to query vector
       . set w=$zzscanalnum
       . if w="" break
# terms
       . set w1=$zstem(w)
# unknown ?
       . if '$data(^dict(w1)) write ?4,"unknown word: ",w,! quit
       . write !,"query term: ",w,!
# stems
       . if $data(^stem(w1)) do
       .. write !,?4,"Words in this stem group are: "
       .. do stems(w1)
       . else  write ?4,w," has no stem related words"
# related
       . write !,?4,"adding related phrase words: "
       . if '$data(^phrase(w1)) do
       .. write "none",!
       . else  do
       .. set f=1
       .. for prs=$order(^phrase(w1,prs)) do
       ... write prs," " set f=0
       ... if $x>60 w !,?26
       ... set ^query(prs)=1
       .. if f write "none"
       .. write !
# soundex
       . set s=$zzsoundex(w1)
       . write ?4,"words that may sound similar to ",w,": "
       . if '$data(^sndx(s)) write "none."
       . else  do
       .. set f=1
       .. for ws=$order(^sndx(s,ws)) do
       ... if $e(w1,1,5)=$e(ws,1,5)&(w1'=ws) do
       .... set f=0
       .... do stems(ws)
       .. if f write "none"
       . if $x'=26 write !
# query word
       . set ^query(w)=3
```

```
# Find documents containing one or more query terms.

      . for d=$order(^index(w,d)) set ^tmp(d)=""  // retain doc id

      set time0=$zd1
      write !

      set file=$j_".tmp"
      open 1:file_",new"

      set f=0

      for i=$order(^tmp(i)) do  // calculate cosine between query and each doc
      . set c=$zzcosine(^doc(i),^query)   // MDH cosine calculation
      . if c>0 use 1 write $justify(c,5,3)," ",i,! set f=f+1

      close 1
      use 5
      if 'f halt

      shell sort -nr < &~file~ > &~"sorted"_file~

      open 1:"sorted"_file_",old"

      use 5
      write f," documents found.",!!
      write "   Doc Rel   Title",!

      for i=1:1:10 do   // display loop
      . use 1 read a
      . if '$test break
      . set d=$piece(a," ",2)
      . set c=$piece(a," ",1)
      . use 5 write $justify(d,6)," ",c," ",^title(d),!
      . write ?14,"matched keys: "
      . for w=$order(^doc(d,w)) do
      .. if $data(^query(w)) do
      ... if $data(^stem(w)) do
      .... for w1=$order(^stem(w,w1)) write w1," "
      ... else  write "*",w," " write:$x>60 ?26
      . write !

      use 5
      close 1

      write !,"*Time used: ",$zd1-time0," seconds",!
       halt
```

```
stems(ws)
       for w3=$order(^stem(ws,w3)) write:$x>60 ?26 write w3," "
       quit
```

Program 34 *medlineRetrieve.mps*

An example of the output of Program 34 is given in Figure 69 where the input query was:

```
staphylococcus aureus antibiotics
```

An asterisk in the *matched keys* section indicates the term is not an expanded word stem. Absence of an asterisk means the term shown is one of the terms that reduced to a matching word stem.

```
query term: staphylococcus

   Words in this stem group are: staphylococcus
   adding related phrase words: isolate-staphylococcus methicillin-resistant-staphylococcus
                         organism-staphylococcus staphylococcus-aureus
                         staphylococcus-endocarditis staphylococcus-epidermidis
                         staphylococcus-infections staphylococcus-strain

   words that may sound similar to staphylococcus: staphylococcal
                         staphylococci

query term: aureus

   Words in this stem group are: aureus
   adding related phrase words: aureus-endocarditis aureus-epidermidis
                         aureus-infections aureus-strain fibronectin-aureus
                         isolate-aureus methicillin-aureus methicillin-resistant-aureus
                         organism-aureus s-aureus staphylococcus-aureus

   words that may sound similar to aureus: none

query term: antibiotics

   Words in this stem group are: antibiotic antibiotics
   adding related phrase words: abscess-antibiotic aminoglycoside-antibiotic
                         antibiotic-bacteremia antibiotic-beta-lactam
                         antibiotic-catheter antibiotic-efficacies
                         antibiotic-infections antibiotic-organism
                         antibiotic-parenteral antibiotic-prophylaxis
                         antibiotic-regimen antibiotic-resistant
                         antibiotic-solution antibiotic-topical
                         appropriate-antibiotic bacterial-antibiotic
                         cephalosporin-antibiotic debride-antibiotic
                         drainage-antibiotic endocarditis-antibiotic
                         fever-antibiotic isolate-antibiotic
                         prophylactic-antibiotic strain-antibiotic
                         susceptibilities-antibiotic susceptibility-antibiotic
                         valve-antibiotic vancomycin-antibiotic
                         wound-antibiotic
   words that may sound similar to antibiotics: antibacterial
                         antibacterials antibody-coated antibody-dependent
                         antibody-mediated antibody-positive

155 documents found.

   Doc Rel   Title
 18728 0.308 Intrinsic methicillin resistance and phage complex of Staphylococcus aureus
             matched keys: aureus *methicillin-aureus staphylococcus *staphylococcus-aureus
 21989 0.302 Toxic shock syndrome associated with Staphylococcus aureus enterocolitis
             matched keys: aureus staphylococcus *staphylococcus-aureus

  7048 0.299 Evolution of the hyperimmunoglobulin E and recurrent infection HIE  JOB's
             syndrome in a youn
             matched keys: aureus *s-aureus staphylococcus *staphylococcus-aureus

 22953 0.298 High rate of methicillin resistance of Staphylococcus aureus isolated from
             hospitalized nursin
             matched keys: aureus *aureus-strain *isolate-aureus
                         *isolate-staphylococcus *methicillin-aureus
                         *methicillin-resistant-aureus *methicillin-resistant-staphylococcus
staphylococcus *staphylococcus-aureus
                         *staphylococcus-strain

  5676 0.290 Single and combination-antibiotic therapy for experimental endocarditis caused
             by methicillin-
             matched keys: aureus *methicillin-resistant-aureus
                         *methicillin-resistant-staphylococcus
                         staphylococcus *staphylococcus-aureus

 13303 0.276 Role of beta-lactamase in expression of resistance by methicillin-resistant
             Staphylococcus aur
             matched keys: *antibiotic-beta-lactam antibiotic-resistant aureus
                           *aureus-strain
                           *isolate-antibiotic *isolate-aureus
                           *isolate-staphylococcus *methicillin-resistant-aureus
                           *methicillin-resistant-staphylococcus
                           staphylococcus *staphylococcus-aureus
                           *staphylococcus-strain *strain-antibiotic

 11616 0.268 Phenotypic expression and genetic heterogeneity of lincosamide inactivation in
             Staphylococcus
             matched keys: aureus *aureus-strain *isolate-aureus
                           *isolate-staphylococcus *s-aureus staphylococcus
                           *staphylococcus-aureus
                           *staphylococcus-strain

 10351 0.257 IgE antibodies to Staphylococcus aureus in the hypereosinophilic syndrome
             matched keys: aureus staphylococcus *staphylococcus-aureus

  5084 0.251 Antibiotic treatment of Staphylococcus aureus endocarditis A review of   cases
             matched keys: *aminoglycoside-antibiotic *antibiotic-beta-lactam
                           *antibiotic-regimen aureus *aureus-strain
                           staphylococcus *staphylococcus-aureus
                           *staphylococcus-strain *strain-antibiotic
 10889 0.249 Acute crescentic glomerulonephritis as a complication of a Staphylococcus
             aureus abscess of hi
             matched keys: aureus staphylococcus *staphylococcus-aureus


*Time used: 0 seconds
```

Figure 69 Enhanced retrieval output

13.2 Controlled Vocabulary Retrieval

Hierarchical indexing schemes such as MeSH (Medical Subject Headings), the Library of Congress Classification System, the ACM's Computing Classification System, the Open Directory Project, and many others are widely used to organize and access information. Thus, we begin with some techniques to manipulate hierarchies. For the most part, these begin with a controlled vocabulary maintained by experts.

13.2.1 The Medical Subject Headings (MeSH)

MeSH (Medical Subject Headings) is a controlled vocabulary hierarchical indexing and classification system developed by the National Library of Medicine (NLM). The MeSH codes are used to code medical records and literature as part of an ongoing research project at the NLM.

The following examples make use of the 2003 MeSH Tree Hierarchy. Newer versions, essentially similar to these, are available from NLM.

Note: *for clinical purposes, the copy of the MeSH hierarchy used here is out of date and should not be employed for clinical decision making. It is used here purely as an example to illustrate a hierarchical index*[31]*.*

The 2003 MeSH file contains approximately 40,000 entries. Each line consists of text along with hierarchical codes describing the subject heading. Figure 70 contains a sample from the 2003 MeSH file.

31 Warning language required by NLM.

```
Body Regions;A01
Abdomen;A01.047
Abdominal Cavity;A01.047.025
Peritoneum;A01.047.025.600
Douglas' Pouch;A01.047.025.600.225
Mesentery;A01.047.025.600.451
Mesocolon;A01.047.025.600.451.535
Omentum;A01.047.025.600.573
Peritoneal Cavity;A01.047.025.600.678
Retroperitoneal Space;A01.047.025.750
Abdominal Wall;A01.047.050
Groin;A01.047.365
Inguinal Canal;A01.047.412
Umbilicus;A01.047.849
Back;A01.176
Lumbosacral Region;A01.176.519
Sacrococcygeal Region;A01.176.780
Breast;A01.236
Nipples;A01.236.500
Extremities;A01.378
Amputation Stumps;A01.378.100
```

Figure 70 Sample MeSH Hierarchy

The format of the MeSH table is:

1. a short text description
2. a semi-colon, and
3. a sequence of decimal point separated codes.

An example of the tree structure thus defined can be seen in Figure 72. The hierarchical codes classify and categorize the data. For example, *A01* and its sub codes are all *body regions; A02* and its descendants concern the *Musculoskeletal System*; *A03* is the top level descriptor for the *Digestive System* and so forth. The codes beneath each of these sub-divide the top level descriptor. Thus, for *A01* we have second level codes such as *A01.047* the Abdomen division; *A01.176* for the Back; *A01.236* for the Breast and so forth. Each of these is further subdivided to a finer levels of detail.

The MeSH codes are an example of a *controlled vocabulary*. That is, a collection of indexing terms that are preselected, defined and authorized by an authoritative source.

13.2.2 Building a MeSH Structured Global Array

The goal here is to write a program that will build a global array tree whose structure corresponds to the MeSH hierarchy. In this tree, each successive index in a global array reference will be a successive code from the 2003 MeSH hierarchy. The text part of each MeSH entry will be stored as the global array data value.

To do this, we want to write a program consisting of Mumps assignment statements similar to the fragment shown in Figure 71. In this example, the code identifiers from the MeSH hierarchy become global array indices and the corresponding text becomes assigned values.

```
set ^mesh("A01")="Body Regions"
set ^mesh("A01","047")="Abdomen"
set ^mesh("A01","047","025")="Abdomenal Cavity"
set ^mesh("A01","047","025","600")="Peritoneum"
.
.
.
set ^mesh("A01","047","365")="Groin"
.
.
.
```

Figure 71 Global Array Commands

A graphical representation of a portion of the MeSH hierarchy can be seen in Figure 72 which depicts the MeSH tree and the corresponding Mumps assignment statements used to create the global array corresponding to the diagram.

A program to build a MeSH tree is shown in Program 35. However, rather than being a program consisting of several thousand Mumps assignment statements, instead we use the Mumps indirection facility to write a short Mumps program that reads the MeSH file and dynamically generates and executes several thousand assignment statements.

The program in Program 35, in a loop (lines 7 through 37), reads a line from the file *mesh2003.txt* (line 9). On lines 11 and 12 the part of the MeSH entry prior to and following the semi-colon are extracted into the strings *key* and *code*, respectively. The loop on lines 15 through 17 extracts each decimal point separated element of *code* into successively numbered elements of the local array *x*. On line 21 a string is assigned to the variable *z*. This will be the initial portion of the global array reference to be constructed.

On line 28 elements of the array *x* are concatenated onto *z* with encompassing quotes and separating commas. On line 29 the final element of array *x* is added along with a closing parenthesis, an assignment operator and the value of *key* and the text are prepended with a Mumps *set* command.

Now the contents of *z* look like a Mumps assignment statement and this is executed on line 37 thus creating an entry in the database. The *xecute* command in Mumps causes the string passed to it to be executed as Mumps code.

Note that to embed a double-quote character (") into a string, you place two immediately adjacent double-quote characters into the string. Thus: """" means a string of length one containing a single double-quote character.

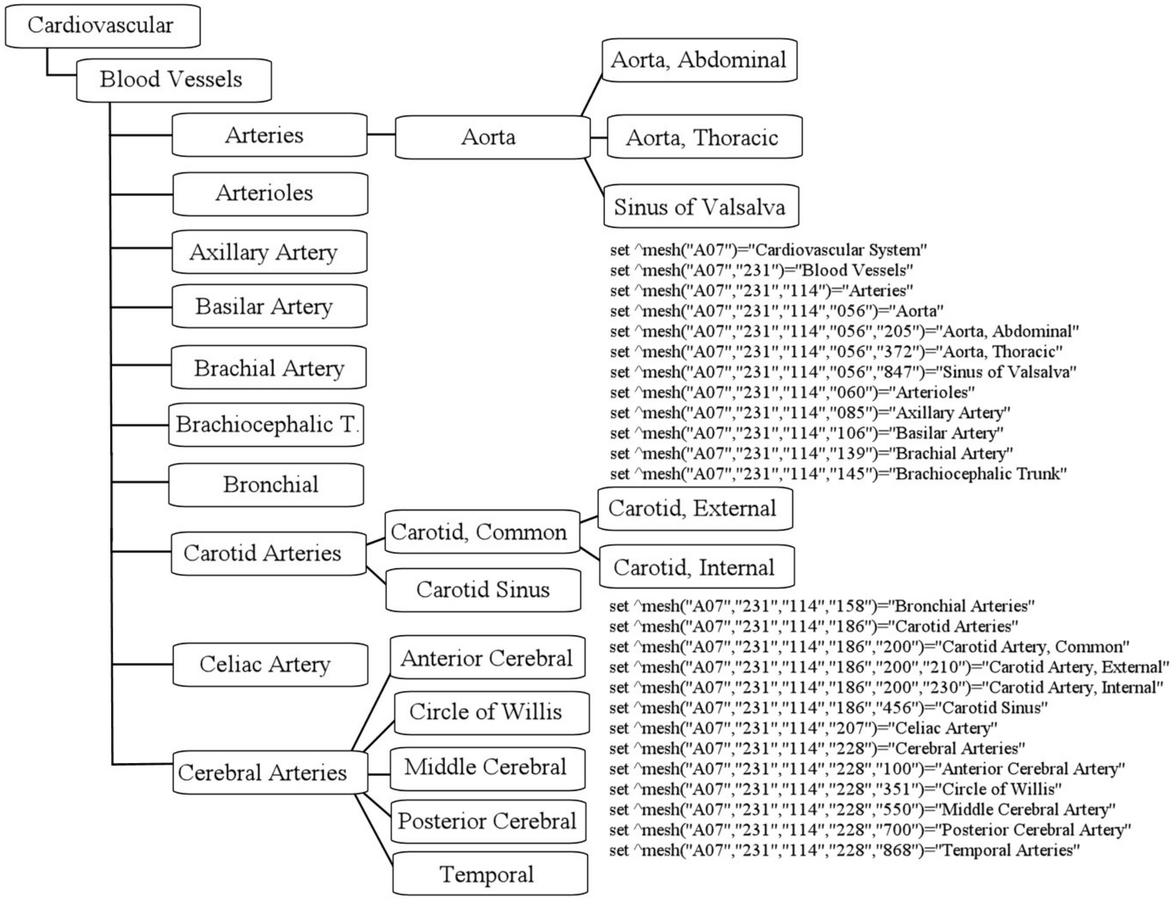
Also note that line 13 uses the OR operator (!) to test if either *key* or *code* is the empty string. Observe that parentheses are needed in this expression since expressions in Mumps are executed left-to-right without precedence. Without parentheses, the predicate would evaluate as if it had been written as:

```
(((key="")!code)="")
```

which would yield a completely different result!

Line 28 uses the concatenation operator (_) on the local array `x(j)`. Local arrays should be used as little as possible as access to them through the Mumps run-time symbol table which can be slow especially if there are a large number of variables or array elements in the current program.

The *close* command on line 39 releases the file associated with unit 1 and makes unit 1 available for re-use. Closing a file opened for input is not strictly needed unless you want to reuse the unit number. Closing a file open for output, however, is desirable in order to flush the internal system buffers to disk. If the program crashes before an output file is closed, it is possible to lose data.



```
set ^mesh("A07")="Cardiovascular System"
set ^mesh("A07","231")="Blood Vessels"
set ^mesh("A07","231","114")="Arteries"
set ^mesh("A07","231","114","056")="Aorta"
set ^mesh("A07","231","114","056","205")="Aorta, Abdominal"
set ^mesh("A07","231","114","056","372")="Aorta, Thoracic"
set ^mesh("A07","231","114","056","847")="Sinus of Valsalva"
set ^mesh("A07","231","114","060")="Arterioles"
set ^mesh("A07","231","114","085")="Axillary Artery"
set ^mesh("A07","231","114","106")="Basilar Artery"
set ^mesh("A07","231","114","139")="Brachial Artery"
set ^mesh("A07","231","114","145")="Brachiocephalic Trunk"
set ^mesh("A07","231","114","158")="Bronchial Arteries"
set ^mesh("A07","231","114","186")="Carotid Arteries"
set ^mesh("A07","231","114","186","200")="Carotid Artery, Common"
set ^mesh("A07","231","114","186","200","210")="Carotid Artery, External"
set ^mesh("A07","231","114","186","200","230")="Carotid Artery, Internal"
set ^mesh("A07","231","114","186","456")="Carotid Sinus"
set ^mesh("A07","231","114","207")="Celiac Artery"
set ^mesh("A07","231","114","228")="Cerebral Arteries"
set ^mesh("A07","231","114","228","100")="Anterior Cerebral Artery"
set ^mesh("A07","231","114","228","351")="Circle of Willis"
set ^mesh("A07","231","114","228","550")="Middle Cerebral Artery"
set ^mesh("A07","231","114","228","700")="Posterior Cerebral Artery"
set ^mesh("A07","231","114","228","868")="Temporal Arteries"
```

Figure 72 MeSH Tree

```
#!/usr/bin/mumps
```

```
# BuildMeshTree.mps

      kill ^mesh
      open 1:"mtrees2003.txt,old"
      if '$test write "mtrees2003.txt not found",! halt
      for  do
      . use 1
      . read a
      . if '$test break
      . set key=$piece(a,";",1)  // text description
      . set code=$piece(a,";",2) // everything else
      . if key=""!(code="") break

      . for i=1:1 do
      .. set x(i)=$piece(code,".",i)  // extract code numbers
      .. if x(i)="" break

      . set i=i-1
      . use 5
      . set z="^mesh("        // begin building a global reference

#-----------------------------------------------------------------------
#     build a reference like ^mesh("A01","047","025","600)
#     by concatenating quotes, codes, quotes, and commas onto z
#-----------------------------------------------------------------------

      . for j=1:1:i-1 set z=z_""""_x(j)_""","
      . set z="set "_z_""""_x(i)_""")="""_key_""""

#-----------------------------------------------------------------------
#     z now looks like set ^mesh("A01","047")="Abdomen"
#     now execute the text
#-----------------------------------------------------------------------

      . write z,!
      . xecute z

      close 1
      use 5
      write "done",!
      halt
```

Program 35 MeSH Structured Global Array

The output of Program 35 is shown in Figure 73. Line 36 writes the text of the created mumps *set* command. These are the commands executed by the *xecute* command on line 37.

Displaying the MeSH Global Array Part I

Now that the MeSH global array has been created, the question is, how to print it properly indented so as to show the tree structure of the data.

Program 36 gives one way to print the global array and the results are shown in Figure 74. In this example there are nested loops to print data at lower levels. When data at a given level is printed, it is indented by 0, 5, 10, and 15 spaces depending on the level of the data.

On Line 4[32] the process begins by finding successive values of the first index of ^*mesh*. Each iteration of this outermost loop will yield, in alphabetic order, a new top level value until there are none remaining. These are placed in the local variable *lev1.*

32 The format of this *for* command uses a non-standard Mumps syntax extension.

```
set ^mesh("A01")="Body Regions"
set ^mesh("A01","047")="Abdomen"
set ^mesh("A01","047","025")="Abdominal Cavity"
set ^mesh("A01","047","025","600")="Peritoneum"
set ^mesh("A01","047","025","600","225")="Douglas' Pouch"
set ^mesh("A01","047","025","600","451")="Mesentery"
set ^mesh("A01","047","025","600","451","535")="Mesocolon"
set ^mesh("A01","047","025","600","573")="Omentum"
set ^mesh("A01","047","025","600","678")="Peritoneal Cavity"
set ^mesh("A01","047","025","750")="Retroperitoneal Space"
set ^mesh("A01","047","050")="Abdominal Wall"
set ^mesh("A01","047","365")="Groin"
set ^mesh("A01","047","412")="Inguinal Canal"
set ^mesh("A01","047","849")="Umbilicus"
set ^mesh("A01","176")="Back"
set ^mesh("A01","176","519")="Lumbosacral Region"
set ^mesh("A01","176","780")="Sacrococcygeal Region"
set ^mesh("A01","236")="Breast"
set ^mesh("A01","236","500")="Nipples"
set ^mesh("A01","378")="Extremities"
set ^mesh("A01","378","100")="Amputation Stumps"
set ^mesh("A01","378","610")="Lower Extremity"
set ^mesh("A01","378","610","100")="Buttocks"
set ^mesh("A01","378","610","250")="Foot"
set ^mesh("A01","378","610","250","149")="Ankle"
set ^mesh("A01","378","610","250","300")="Forefoot, Human"
set ^mesh("A01","378","610","250","300","480")="Metatarsus"
 .
 .
 .
```

Figure 73 Creating the Mesh tree

The program then advances to line 6 which yields successive values of all second level codes subordinate to the value of the current top level code (*lev1*). Each of these is placed in *lev2*. The second level codes are printed on line 7 indented by 5 spaces.

```
#!/usr/bin/mumps
 # BasicMtreePrint.mps

        for lev1=$order(^mesh(lev1)) do
        . write lev1," ",^mesh(lev1),!
        . for lev2=$order(^mesh(lev1,lev2)) do
        .. write ?5,lev2," ",^mesh(lev1,lev2),!
        .. for lev3=$order(^mesh(lev1,lev2,lev3)) do
        ... write ?10,lev3," ",^mesh(lev1,lev2,lev3),!
         ... for lev4=$order(^mesh(lev1,lev2,lev3,lev4)) do
        .... write ?15,lev4," ",^mesh(lev1,lev2,lev3,lev4),!
```

Program 36 Print the MeSH tree

The process continues for levels 3 and 4. If there are no codes at a given level, the loop at that level terminates immediately and flow is returned to the outer loop and any more deeply nested loops are not executed.

```
A01 Body Regions
     047 Abdomen
          025 Abdominal Cavity
               600 Peritoneum
               750 Retroperitoneal Space
          050 Abdominal Wall
          365 Groin
          412 Inguinal Canal
          849 Umbilicus
     176 Back
          519 Lumbosacral Region
          780 Sacrococcygeal Region
     236 Breast
          500 Nipples
     378 Extremities
          100 Amputation Stumps
          610 Lower Extremity
               100 Buttocks
               250 Foot
               400 Hip
               450 Knee
               500 Leg
               750 Thigh
          800 Upper Extremity
               075 Arm
               090 Axilla
               420 Elbow
               585 Forearm
               667 Hand
          750 Shoulder
     456 Head
          313 Ear
          505 Face
               173 Cheek
               259 Chin
               420 Eye
               580 Forehead
               631 Mouth
               733 Nose
               750 Parotid Region
          810 Scalp
          830 Skull Base
               150 Cranial Fossa, Anterior
               165 Cranial Fossa, Middle
               200 Cranial Fossa, Posterior
     598 Neck
     673 Pelvis
          600 Pelvic Floor
     719 Perineum
     911 Thorax
          800 Thoracic Cavity
               500 Mediastinum
               650 Pleural Cavity
          850 Thoracic Wall
     960 Viscera
A02 Musculoskeletal System
     165 Cartilage
          165 Cartilage, Articular
          207 Ear Cartilages
          410 Intervertebral Disk
          507 Laryngeal Cartilages
               083 Arytenoid Cartilage
               211 Cricoid Cartilage
               411 Epiglottis
               870 Thyroid Cartilage
          590 Menisci, Tibial
          639 Nasal Septum
     340 Fascia
          424 Fascia Lata
     513 Ligaments
          170 Broad Ligament
          514 Ligaments, Articular
               100 Anterior Cruciate Ligament
               162 Collateral Ligaments
               287 Ligamentum Flavum
               350 Longitudinal Ligaments
               475 Patellar Ligament
```

```
            600 Posterior Cruciate Ligament
      ...
```

Figure 74 Printed the Mesh tree

13.2.2.1 Printing the MeSH Global Array Part II

Program 37 presents a more general function to print the *^mesh* hierarchy and its output is shown in Figure 75. The code in Program 37 works with trees of any depth whereas the code in program 36 needs to be extended in order to work with trees whose depth is greater than four.

```
1  #!/usr/bin/mumps
2  # AdvancedMtreePrint.mps
3
4         set x="^mesh"
5         for  do
6         . set x=$query(@x)
7         . if x="" break
8         . set i=$qlength(@x)
9         . write ?i*2," ",$qsubscript(x,i)," ",@x,?50,x,!
```

Program 37 Alternate MeSH tree printing program

In the example in Program 37, we first set a local variable *x* to *^mesh*, the name of the MeSH global array. In the loop on lines 5 through 9, the variable *x* is passed as an argument to the builtin function *$query()* which returns the next ascendant global array key in the database. These are printed in line 9 and can be seen in the right hand column output of Program 37 shown in Figure 75. The values returned by *$query()* are assigned to the variable *x* and used as the seed to *$query()* in the next iteration unto, finally, an empty string is returned.

In line 8 the number of subscripts in the global array reference contained in variable *x* is assigned to the local variable *i*. In line 9 this number is used to indent the output by twice the number of spaces as there are subscripts (*?i*2*).

On line 9, after indenting, we want to print the MeSH code value for the level and the text followed by, in a separate column, the full Mumps *^mesh()* global array reference.

We use the builtin function *$qsubscript()* to extract from the global array reference the i^{th} index element. The arguments to *$qsubscript()* are the text of the global (or local) array *x* and an integer *i* where *i* is the current level of the tree. The function returns the value of the i^{th} subscript.

next we print the value stored at the global array, the text of the description of the MeSH codes. Note that the expression *@x* evaluates the string in variable *x* which, since it is a global array reference, evaluates to the value stored at the global array node reference.

In a column indented to 50 we print the actual MeSH global array reference. Note that the last index of each global array reference is also printed (without quotes) at the beginning of the line. These are the values extracted from the global array references by *$qsubscript()*.

```
A01 Body Regions                          ^mesh("A01")
   047 Abdomen                             ^mesh("A01","047")
     025 Abdominal Cavity                  ^mesh("A01","047","025")
       600 Peritoneum                      ^mesh("A01","047","025","600")
         225 Douglas' Pouch                ^mesh("A01","047","025","600","225")
         451 Mesentery                     ^mesh("A01","047","025","600","451")
           535 Mesocolon                   ^mesh("A01","047","025","600","451","535")
         573 Omentum                       ^mesh("A01","047","025","600","573")
         678 Peritoneal Cavity             ^mesh("A01","047","025","600","678")
       750 Retroperitoneal Space           ^mesh("A01","047","025","750")
     050 Abdominal Wall                    ^mesh("A01","047","050")
     365 Groin                             ^mesh("A01","047","365")
     412 Inguinal Canal                    ^mesh("A01","047","412")
     849 Umbilicus                         ^mesh("A01","047","849")
   176 Back                                ^mesh("A01","176")
     519 Lumbosacral Region                ^mesh("A01","176","519")
     780 Sacrococcygeal Region             ^mesh("A01","176","780")
   236 Breast                              ^mesh("A01","236")
     500 Nipples                           ^mesh("A01","236","500")
   378 Extremities                         ^mesh("A01","378")
     100 Amputation Stumps                 ^mesh("A01","378","100")
     610 Lower Extremity                   ^mesh("A01","378","610")
       100 Buttocks                        ^mesh("A01","378","610","100")
       250 Foot                            ^mesh("A01","378","610","250")
         149 Ankle                         ^mesh("A01","378","610","250","149")
         300 Forefoot, Human               ^mesh("A01","378","610","250","300")
           480 Metatarsus                  ^mesh("A01","378","610","250","300","480")
           792 Toes                        ^mesh("A01","378","610","250","300","792")
             380 Hallux                    ^mesh("A01","378","610","250","300","792","380")
         510 Heel                          ^mesh("A01","378","610","250","510")
       400 Hip                             ^mesh("A01","378","610","400")
       450 Knee                            ^mesh("A01","378","610","450")
       500 Leg                             ^mesh("A01","378","610","500")
       750 Thigh                           ^mesh("A01","378","610","750")
     800 Upper Extremit                    ^mesh("A01","378","800")
       075 Arm                             ^mesh("A01","378","800","075")
       090 Axilla                          ^mesh("A01","378","800","090")
       420 Elbow                           ^mesh("A01","378","800","420")
       585 Forearm                         ^mesh("A01","378","800","585")
       667 Hand                            ^mesh("A01","378","800","667")
         430 Fingers                       ^mesh("A01","378","800","667","430")
           705 Thumb                       ^mesh("A01","378","800","667","430","705")
         715 Wrist                         ^mesh("A01","378","800","667","715")
       750 Shoulder                        ^mesh("A01","378","800","750")
   456 Head                                ^mesh("A01","456")
     313 Ear                               ^mesh("A01","456","313")
     505 Face                              ^mesh("A01","456","505")
       173 Cheek                           ^mesh("A01","456","505","173")
```

```
      259 Chin                           ^mesh("A01","456","505","259")
      420 Eye                            ^mesh("A01","456","505","420")
        338 Eyebrows                     ^mesh("A01","456","505","420","338")
        504 Eyelids                      ^mesh("A01","456","505","420","504")
          421 Eyelashes                  ^mesh("A01","456","505","420","504","421")
      580 Forehead                       ^mesh("A01","456","505","580")
      631 Mouth                          ^mesh("A01","456","505","631")
        515 Lip                          ^mesh("A01","456","505","631","515")
```

Figure 75 Alternative MeSH printing output

13.2.2.2 Displaying Global Arrays in Key Order

A program to print the global array references in the global array *b-tree* database tree order is shown in Program 38. Here we use the Mumps function *$query()* to access the *b-tree* keys in the order in which they are actually stored, in sequential key order, as shown in Figure 76.

The program shown in Program 38 passes to *$query()* a string containing a global array reference. The function returns the next ascending global array reference in the database system. Eventually, it will run out of ^*mesh* references and return an empty string which causes program termination.

```
1  ^mesh("A01")
2  ^mesh("A01","047")
3  ^mesh("A01","047","025")
4  ^mesh("A01","047","025","600")
5  ^mesh("A01","047","025","600","225")
6  ^mesh("A01","047","025","600","451")
7  ^mesh("A01","047","025","600","451","535")
8  ^mesh("A01","047","025","600","573")
9  ^mesh("A01","047","025","600","678")
10 ^mesh("A01","047","025","750")
11 ^mesh("A01","047","050")
12 ^mesh("A01","047","365")
13 ^mesh("A01","047","412")
14 ^mesh("A01","047","849")
15 ^mesh("A01","176")
```

Figure 76 MeSH global array codes

Note that the line:

```
. write x,?50,@x,!
```

displays the global array reference in variable *x* and then prints the contents of the node at *x* by evaluating the global array reference (@x). Evaluation of a variable by the indirection operator yields the value of the variable.

```
#!/usr/bin/mumps
# AdvancedMtreePrint2.mps

      set x="^mesh"  // build the first index
      for  do
      . set x=$query(@x) // get next array reference
      . if x="" break
      . write x,?50,@x,!
```

Program 38 Program to print MeSH global

The output from Program 38 appears in Figure 77.

```
^mesh("A01")                                     Body Regions
^mesh("A01","047")                               Abdomen
^mesh("A01","047","025")                         Abdominal Cavity
^mesh("A01","047","025","600")                   Peritoneum
^mesh("A01","047","025","600","225")             Douglas' Pouch
^mesh("A01","047","025","600","451")             Mesentery
^mesh("A01","047","025","600","451","535")       Mesocolon
^mesh("A01","047","025","600","573")             Omentum
^mesh("A01","047","025","600","678")             Peritoneal Cavity
^mesh("A01","047","025","750")                   Retroperitoneal Space
^mesh("A01","047","050")                         Abdominal Wall
^mesh("A01","047","365")                         Groin
^mesh("A01","047","412")                         Inguinal Canal
^mesh("A01","047","849")                         Umbilicus
^mesh("A01","176")                               Back
^mesh("A01","176","519")                         Lumbosacral Region
^mesh("A01","176","780")                         Sacrococcygeal Region
^mesh("A01","236")                               Breast
^mesh("A01","236","500")                         Nipples
^mesh("A01","378")                               Extremities
^mesh("A01","378","100")                         Amputation Stumps
^mesh("A01","378","610")                         Lower Extremity
^mesh("A01","378","610","100")                   Buttocks
^mesh("A01","378","610","250")                   Foot
^mesh("A01","378","610","250","149")             Ankle
^mesh("A01","378","610","250","300")             Forefoot, Human
^mesh("A01","378","610","250","300","480")       Metatarsus
^mesh("A01","378","610","250","300","792")       Toes
^mesh("A01","378","610","250","300","792","380") Hallux
^mesh("A01","378","610","250","510")             Heel
^mesh("A01","378","610","400")                   Hip
^mesh("A01","378","610","450")                   Knee
^mesh("A01","378","610","500")                   Leg
^mesh("A01","378","610","750")                   Thigh
^mesh("A01","378","800")                         Upper Extremity
^mesh("A01","378","800","075")                   Arm
^mesh("A01","378","800","090")                   Axilla
^mesh("A01","378","800","420")                   Elbow
^mesh("A01","378","800","585")                   Forearm
^mesh("A01","378","800","667")                   Hand
^mesh("A01","378","800","667","430")             Fingers
^mesh("A01","378","800","667","430","705")       Thumb
^mesh("A01","378","800","667","715")             Wrist
```

```
^mesh("A01","378","800","750")                    Shoulder
```

Figure 77 MeSH global printed

13.2.2.3 Searching the MeSH Global Array

Next we want to write a program that will, when given a keyword, locate all the MeSH headings containing the keyword and display the full heading, hierarchy codes, and descendants of the keywords found at this level. In effect, this program retrieves all the more specific terms related to a higher level, more general term. The program is shown in Program 39.

The program in Program 39 first reads in a keyword into a local variable *key.* It next assigns to local variable *x* the text of the initial element of the global array reference *^mesh.*

It now iterates through *^mesh* global array references (until there are none remianing) in the loop in lines 9 through 19.

The program examines the value stored for each *^mesh* global array node in global array key order.

On line 10, the *$find()* function determines if the value stored at the current global array node referenced by *x* (The value stored at which is@*x*) contains, as a substring, the value in *key*. That is, for node in the *^mesh* tree, does the text value stored at the node contain *key*?

```
1  #!/usr/bin/mumps
2  # FindMesh.mps December 5, 2011
3
4        read "enter keyword: ",key
5        write !
6        set x="^mesh"  // build a global array ref
7        set x=$query(@x)
8        if x="" halt
9        for  do
10        . if '$find(@x,key) set x=$query(@x) // is key stored at this ref?
11       . else  do
12       .. set i=$qlength(@x)  // number of subscripts
13       .. write x," ",@x,!
14       .. for  do
15       ... set x=$query(@x)
16       ... if x="" halt
17       ... if $qlength(@x)'>i break
18       ... write ?5,x," ",@x,!
19       . if x="" halt
```

Program 39 Program to search MeSH global array

If *$find()* does not detect the value in *key* in the global array node, the next global array reference is returned by *$query()* until there are none remaining.

```
enter keyword: Skeleton
^mesh("A02","835") Skeleton
^mesh("A02","835","232") Bone and Bones
^mesh("A02","835","232","087") Bones of Upper Extremity
^mesh("A02","835","232","087","144") Carpal Bones
^mesh("A02","835","232","087","144","650") Scaphoid Bone
^mesh("A02","835","232","087","144","663") Semilunar Bone
^mesh("A02","835","232","087","227") Clavicle
^mesh("A02","835","232","087","412") Humerus
^mesh("A02","835","232","087","535") Metacarpus
^mesh("A02","835","232","087","702") Radius
^mesh("A02","835","232","087","783") Scapula
^mesh("A02","835","232","087","783","261") Acromion
^mesh("A02","835","232","087","911") Ulna
^mesh("A02","835","232","169") Diaphyses
^mesh("A02","835","232","251") Epiphyses
^mesh("A02","835","232","251","352") Growth Plate
...
```

Figure 78 MeSH keyword search results

If the *key* is found, however, the program prints the reference and value stored then it scans for additional references whose number of subscripts is greater than that of the found reference (that is, sub trees of the found reference) and it prints any nodes that are subordinate to the found node since these are necessarily more specific (deeper) forms of the term sought.

The function *$qlength()* returns the number of subscripts in a reference. When the number of subscripts becomes *less-than-or-equal* (shown as[33] *not-greater-than*: '>) to the number of subscripts in the found reference, printing ends and the key scan of the nodes resumes. Thus, only sub-trees of the found node will be printed.

After all the nodes of greater depth are printed, the program looks for additional instances of the search keyword.

An example is shown in Figure 78 where *Skeleton* was given as input.

33 Mumps does not have *greater-than-or-equal* or *less-than-or-equal* operators so we use *not-less-than* and *not-greater-than*, respectively, instead.

13.2.2.4 Display OHSUMED Collection by Embedded MeSH Headings

Next, we write a program to read the MEDLINE formatted abstracts (from the modified TREC-9 data base described above) and write out a list of MeSH headings, the number of times each heading occurs, and the title of each abstract in which it occurs along with the byte offset of the abstract in the master file.

This is an example of an inverted index using a controlled vocabulary. That is, a mapping from a collection of pre-existing index terms, in this case the MeSH headings, to the underlying documents containing these headings. An inverted index is faster than sequentially searching each document for index terms.

First note that the lines containing MeSH headings in the OHSUMED data base all have the code *MH*[34] in positions 1 and 2. Note also that there is a blank line that signals the end of each abstract and he beginning of the next one (or the end of file).

Creation of the inverted index proceeds as follows: first, we locate and extract the MeSH terms in the OHSUMED file. Then, for each instance of a MeSH term, we record the term and the offset into the OHSUMED file of the article in which the term occurred in a global array (*^MH*). Additionally, we count the number of times each term occurs. The program to do this is shown in Program 40.

Finally, after the entire OHSUMED file has been processed, we write out each MeSH heading, the number of times it occurs and a list of the titles and their offsets in which it occurred. An example of the output can be seen in Figure 79.

Program 40 opens the input file (line 5), captures the initial file byte offset (line 12) and then loops reading lines from the input file designated as unit 1. The loop ends when there is no more input on unit 1.

34 See Figure 24 MEDLINE Codes on page 51.

```
#!/usr/bin/mumps

# MeshIndex.mps

      open 1:"ohsu.medline,old"
      use 1

      kill ^MH

      set x=0  // a counter to limit the size

      set i=$ztell // return the integer offset in the file

      for  do
      . use 1
      . read a
      . if '$test break

# if a blank line, record the offset - this is the start of an abstract

      . if a="" set i=$ztell set x=x+1 quit  // return the offset in the file

      . if $extract(a,1,3)="MH " do
      .. use 5
      .. set a=$piece($extract(a,7,255),"/",1)

# create or increment entry for word

      .. if $data(^MH(a)) set ^MH(a)=^MH(a)+1
      .. else  set ^MH(a)=1

# store the offset

      .. set ^MH(a,i)=""

# write for each heading the titles associated with it

      use 5
      set x=""
      for  do
      . set x=$order(^MH(x))
      . if x="" break
      . write x," occurs in ",^MH(x)," documents",!
      . for off=$order(^MH(x,off)) do
      .. use 1
      .. do $zseek(off)
      .. for  do
      ... read a
```

```
49       ... if $extract(a,1,3)'="TI " quit
50       ... use 5
51       ... write ?5,off,?15,$extract(a,7,80),!
52       ... break
```

Program 40 Locate instances of MeSH keywords

If an empty line is detected (line 21), the offset is recorded in the local variable *i*, the abstract count *x* is incremented. Note that the value returned by *$ftell()* on line 21 is the byte offset of the line we are *about* to be read, not the one most recently read. Thus, the value in variable *i* is the address of the first line of the *next* abstract.

```
Abdominal Injuries occurs in 13 documents
    1650173  Percutaneous transcatheter steel-coil embolization of a large proximal pos
    1678059  Features of 164 bladder ruptures.
    2523966  Injuries to the abdominal vascular system: how much does aggressive resusc
    3436121  Triple-contrast computed tomography in the evaluation of penetrating poste
    4624903  Correlations of injury, toxicology, and cause of death to Galaxy Flight 20
    4901771  Selective management of blunt abdominal trauma in children--the triage rol
    4913645  Percutaneous peritoneal lavage using the Veress needle: a preliminary repo
    6713150  The seat-belt syndrome.
    7019763  Early diagnosis of shock due to pericardial tamponade using transcutaneous
    7885247  The incidence of severe trauma in small rural hospitals.
    8189154  Intussusception following abdominal trauma.
    8808690  Hepatic and splenic injury in children: role of CT in the decision for lap
    8961708  Peritoneal lavage and the surgical resident.
Abdominal Neoplasms occurs in 6 documents
    10033669 Current spectrum of intestinal obstruction.
    10399042 Diagnosis of metastases from testicular germ cell tumours using fine needl
    116380   Intracystic injection of OK-432: a new sclerosing therapy for cystic hygro
    5804499  Pheochromocytoma, polycythemia, and venous thrombosis.
    8983032  Malignant epithelioid peripheral nerve sheath tumor arising in a benign sc
    8991187  DTIC therapy in patients with malignant intra-abdominal neuroendocrine tum
Abdominal Wall occurs in 11 documents
    10291646 Structure of abdominal muscles in the hamster: effect of elastase-induced
    2142543  Surgical incision for cesarean section.
    2230059  Exstrophy, epispadias, and cloacal and urogenital sinus abnormalities.
    2963791  Adductor tendinitis and musculus rectus abdominis tendopathy.
    5426490  Postpartum sit-ups [letter]
    5438957  Bilateral upper-quadrant (intercostal) flaps: the value of protective sens
    6012451  Anterior rectus sheath repair for inguinal hernia.
    6557458  Effects of upper or lower abdominal surgery on diaphragmatic function.
    8946400  Patterns of muscular activity during movement in patients with chronic low
    8947451  Trunk muscle balance and muscular force.
    9892904  Venous plasma (total) bupivacaine concentrations following lower abdominal
```

Figure 79 Titles organized by MeSH code

Line 23 checks to see if a line contains the code *MH*. If it does, it extracts the portion of the line from position 7 up to, but not including, any / character (we ignore any text following the / character). If no / character is present, we extract to the end of line. The line length limit of 255 is overly generous as no line is that long. The actual length of the MeSH heading stored in local variable *a* is determined by the line length, not 255.

Next the MeSH heading, a pound sign and the offset of where the abstract containing the MeSH term are written to standard output.

For each MeSH heading detected, an instance of the global array ^*MH* is instantiated if it does not exists and the count stored at ^*MH* is incremented *or set to one* (lines 29 and 30). The offset of the document containing it is also recorded in ^*MH* as second level index value (line 34). Thus the global array ^*MH* contains at the first indexing level the MeSH headings and, at the second level, the offset addresses of thos documents that contain the term.

When the input is exhausted, the program prints for each heading the number of documents it appeared in along with a list of the documents. A sample of the output is given in Figure 79. This form of display is called a *concordance* - a list of words and an indication of their location and context[35].

35 Note that MeSH terms are assigned to abstracts by human indexers and the actual abstracts, although they contain the concept indicated by the MeSH term, may not contain the actual term.

```
#!/usr/bin/mumps
# MeshTitles.mps

 open 1:"ohsu.medline,old"
 if '$test write "file open error",! halt

 set x="^mesh(0)"
 for  do
 . set x=$query(x)
 . if x="" break
 . set i=$qlength(x)
 . write ?i*2," ",$qsubscript(x,i)," ",@x,?50,x,!
 . set z=i$zlower(@x)
 . if $data(^MH(z)) do
 .. write !,?i*2+5,z," occurs in ",^MH(z)," documents",!
 .. for off=$order(^MH(z,off)) do
 ... use 1
 ... do $zseek(off)
 ... for  do
 .... read a
 .... if $extract(a,1,3)'="TI " quit
 .... use 5
 .... write ?i*2+5,"  ",$extract(a,7,80),!
 .... break
 .. write !
```

Program 41 Hierarchical MeSH concordance

13.2.2.5 Display of OHSUMED Documents by MeSH Hierarchy

Now we combine the programs from Program 38 (page 200) and Program 40 into a single program that displays the titles integrated into the overall tree structure of the MeSH hierarchy. The code is shown in Program 41 and a sample of the output in Figure 80.

```
       025 Abdominal Cavity                     ^mesh("A01","047","025")
        600 Peritoneum                         ^mesh("A01","047","025","600")

            Peritoneum occurs in 4 documents
              Systems of membranes involved in peritoneal dialysis.
              Suppression of lymphocyte reactivity in vitro by supernatants of explants
              An evaluation of the Gore-Tex surgical membrane for the prevention of post
              The morphologic effect of short-term medical therapy of endometriosis.

          225 Douglas' Pouch                   ^mesh("A01","047","025","600","225")
          451 Mesentery                        ^mesh("A01","047","025","600","451")

              Mesentery occurs in 3 documents
                Cellular localization of angiotensinogen gene expression in brown adipose
                Technique of mesenteric lengthening in ileal reservoir-anal anastomosis.
                Detection of mesenteric involvement in sarcoidosis using computed tomograp

            535 Mesocolon                      ^mesh("A01","047","025","600","451","535")
          573 Omentum                          ^mesh("A01","047","025","600","573")

              Omentum occurs in 5 documents
                The omentum as an untapped reservoir for microvascular conduits.
                Early vascular grafting to prevent upper extremity necrosis after electric
                Evidence for an inhibitor of leucocyte sodium transport in the serum of ne
                Vascular graft seeding [letter]
                Suppression of lymphocyte reactivity in vitro by supernatants of explants

          678 Peritoneal Cavity                ^mesh("A01","047","025","600","678")

              Peritoneal Cavity occurs in 4 documents
                Contribution of lymphatic absorption to loss of ultrafiltration and solute
                Differential expression of the amyloid SAA 3 gene in liver and peritoneal
                The pharmacology of intraperitoneally administered bleomycin.
                Ultrafiltration failure in continuous ambulatory peritoneal dialysis due t

        750 Retroperitoneal Space              ^mesh("A01","047","025","750")

            Retroperitoneal Space occurs in 5 documents
              Failure of adjuvant chemotherapy in testicular cancer.
              Uterine leiomyomas with retroperitoneal lymph node involvement.
              Triple-contrast computed tomography in the evaluation of penetrating poste
              Position of the superior mesenteric artery on computed tomography and its
              Lumbar arterial injury: radiologic diagnosis and management.
```

Figure 80 Hierarchical MeSH concordance

13.2.2.6 Searching By MeSH Terms

An inverted file is organized as a set of keys. For each key there is a set of pointers to those documents in the main document file which are indexed by the key.

For example, consider the program in Program 40 above. In this case the keywords are terms from the MeSH hierarchy found in documents in the OHSUMED collection. These are stored in the global array *^MH(term)* where *term* is an individual MeSH term contained in one or more of the OHSUMED documents.

For each MeSH term entry in *^MH(term)*, there are one or more file offset pointers at the second level of indexing of the *^MH* global array that point to a document in the collection indexed by the term.

Retrieval based on MeSH keyword involves locating the term at the first level of *^MH* and then fetching and displaying each of the corresponding documents whose offsets are at the second level. An example of this along with output is shown in Program 42.

The program in Program 42 reads in a keyword from the user (line 11), prints the number of documents the keyword appears in and then, for each offset recorded at the second level of global array *^MH*, begins reading the original abstract from the *ohsu.medline* file beginning at the offset stored in *^MH*. The program reads through the first through the first few lines of the abstract until it locates the title line which it then prints along with the file offset of the beginning of the document. The total time taken by the program is measured in milliseconds despite the fact that the file being 'searched' is 336 million bytes in length.

```
#!/usr/bin/mumps

# meshword.mps

      open 1:"ohsu.medline,old"
      if '$test write "file open error",! halt

# write for each heading the titles associated with it

      write "Enter a MeSH keyword: "
      read word

      write !,word," occurs in ",^MH(word)," documents",!
      for off=$order(^MH(word,off)) do
      . use 1
      . do $zseek(off)
      . for  do
      .. read a
      .. if $extract(a,1,3)'="TI " quit
      .. use 5
      .. write ?5,off,?15,$extract(a,7,80),!
      .. break

Enter a MeSH keyword: Acetylcholinesterase

Acetylcholinesterase occurs in 6 documents
    141739   The slow channel syndrome. Two new cases.
    2758782  The diagnostic value of acetylcholinesterase/butyrylcholinesterase ratio i
    3643396  Ultrastructural analysis of murine megakaryocyte maturation in vitro: comp
    5479094  Long-term neuropathological and neurochemical effects of nucleus basalis l
    6687870  Cholinesterase activities in cerebrospinal fluid of patients with senile d
    8444730  Increased skeletal muscle acetylcholinesterase activity in porcine maligna
```

Program 42 Inverted search by MeSH term

Clearly, the inverted lookup in Program 42 is preferable to scanning each document looking for instances of the MeSH term *Acetylcholinesterase!*

13.3 KWIC/KWOC Retrieval

In the early days of information retrieval, and still to this day, many indexing schemes have used several simple techniques, based mainly on titles, known as *Key Word In Context* (KWIC), *Key Word Out of Context* (KWOC) or *Key Word Alongside Context* (KWAC) to organize and index content.

Program 43 shows a KWIC program (this may be enabled in *index.script*) and the output is shown in Figure 81. Similarly, Program 44 shows a KWOC program and Figure 82 shows its output.

```
#!/usr/bin/mumps
# kwic.mps Copyright 2014 Kevin C. O'Kane

     set %=$zStopInit("stop.words")

     for  do
     . set i=$zzinput("w")
     . if i=0 break
     . set i=i-1
     . set doc=w(0)
     . for j=1:1:i do
     .. if $l(w(j))<3 quit
     .. if $l(w(j))>24 quit
     .. if w(j)?1n.e quit
     .. set tw=$zlower(w(j))
     .. if $zStopLookup(tw) quit
     .. write tw,?26,doc,": "
     .. set x=""
     .. for k=j+1:1:i set x=x_w(k)_" "
     .. set y=""
     .. for k=j-1:-1:1 set y=w(k)_" "_y quit:$l(y)>25
     .. write ?(80-$l(y)),y,?80,w(j)," ",x,!
```

Program 43 KWIC index

```
269:                    serotonin S2 and thromboxane A2prostaglandin H2 receptor activation activation
976:                       leukocyte scanning of the abdomen Analysis of its value for diagnosis and manageme
118:                               Nonspecific acute abdominal pain letter
426:                                        Eruptive abdominal pain Chylomicronemia
482:                                                 Abdominal and pelvic pain
562:                                       Recurrent abdominal pain as the sole manifestation of hereditary angioedema
779:                                  Anatomy of the abductor muscles of the hip as studied by computed tomography
563:              apolipoprotein B48 in two cases of abetalipoproteinemia
524:                    function and verbal learning ability in patients with complex partial seizures of tempo
```

Figure 81 KWIC output

```
#!/usr/bin/mumps
# kwoc.mps Copyright 2014 Kevin C. O'Kane

      set %=$zStopInit("stop.words")

      for  do
      . set i=$zzinput("w")
      . if i=0 break
      . set doc=w(0)
      . set i=i-1
      . for j=1:1:i do
      .. if $l(w(j))<3 quit
      .. if $l(w(j))>24 quit
      .. if w(j)?1n.e quit
      .. if $zStopLookup(w(j)) quit
      .. write $zlower(w(j)),"*",?25,$justify(doc,6),": "
      .. for k=1:1:i write w(k)," "
      .. write !
```

Program 44 KWOC index

```
abdominal*                  482: Abdominal and pelvic pain
abdominal*                  562: Recurrent abdominal pain as the sole manifestation of hereditary angioedema in
abductor*                   779: Anatomy of the abductor muscles of the hip as studied by computed tomography
abetalipoproteinemia*       563: Absence of intestinal synthesis of apolipoprotein B48 in two cases of abetalipoproteinemia
ability*                    524: Memory function and verbal learning ability in patients with complex partial seizures of
abnormalities*              333: The effects of posture on abnormalities of forearm venous tone in women with
abnormalities*              520: Transient focal abnormalities of neuroimaging studies during focal status epilepticus
abnormality*                 93: A unique 7p12q chromosomal abnormality associated with recurrent abortion and hypofibri
abnormally*                 230: Role of abnormally high transmural pressure in the permselectivity defect of glomerular
abortion*                    93: A unique 7p12q chromosomal abnormality associated with recurrent abortion and hypofibrinog
```

Figure 82 KWOC output

Many journals publish annual KWIC/KWOC indices to help searchers locate articles. These, as shown above, are normally based solely on titles. Unfortunately, not all titles are suitable candidates for indexing as these titles that have appeared on Amazon.com indicate:

- “The Jewish-Japanese Sex and Cook Book and How to Raise Wolves”
- “How To Raise Your I.Q. by Eating Gifted Children”
- “Why Cats Paint”
- “Learning To Play With a Lion’s Testicles”
- “Don’t Bend Over in the Garden, Granny, You Know Them Taters Got Eyes”

- “If God Loves Me, Why Can’t I Get My Locker Open?”
- “I Could Pee on This: And Other Poems by Cats”
- “How to Traumatize Your Children”
- “What If a Lion Eats Me And I Fall Into a Hippopotamus’ Mud Hole?”
- “Stray Shopping Carts of Eastern North America”
- “Extreme Ironing”

13.4 Boolean Retrieval

In a Boolean system documents are retrieved based on a Boolean expression involving keywords or terms in a user provided query.

The sets of documents retrieved as a result of the keywords used are then combined with one another according to Boolean operators such as *AND, OR, NOT, etc*., in order to produce a final set of documents that is presented to the user. For example, the query:

```
COMPUTERS AND (ONCOLOGY OR GASTROENTEROLOGY OR CARDIOLOGY)
```

would produce a set of documents all of which were indexed under the term *COMPUTERS* and at least one or more of the terms ONCOLOGY, GASTROENTEROLOGY or *CARDIOLOGY*.

While many Boolean based systems retrieve a document based on a binary evaluation of whether the documents terms fit the Boolean expression, others score retrieved documents based on factors such as search term frequency and distribution in the collection as a whole[36].

Boolean systems such as these are discussed in Chapter 13.4.1. Figure 83 shows a legacy CAS Boolean based search from 1979.

13.4.1 Boolean Based Searching

Before we examine the Vector Space model we should explore some very basic search techniques because some aspects of these are used with the Vector Space model.

A number of popular commercial information retrieval systems over the years have been based on queries rooted in Boolean logic. In these systems, query terms are typically connected by operators such as *AND*, *OR*,

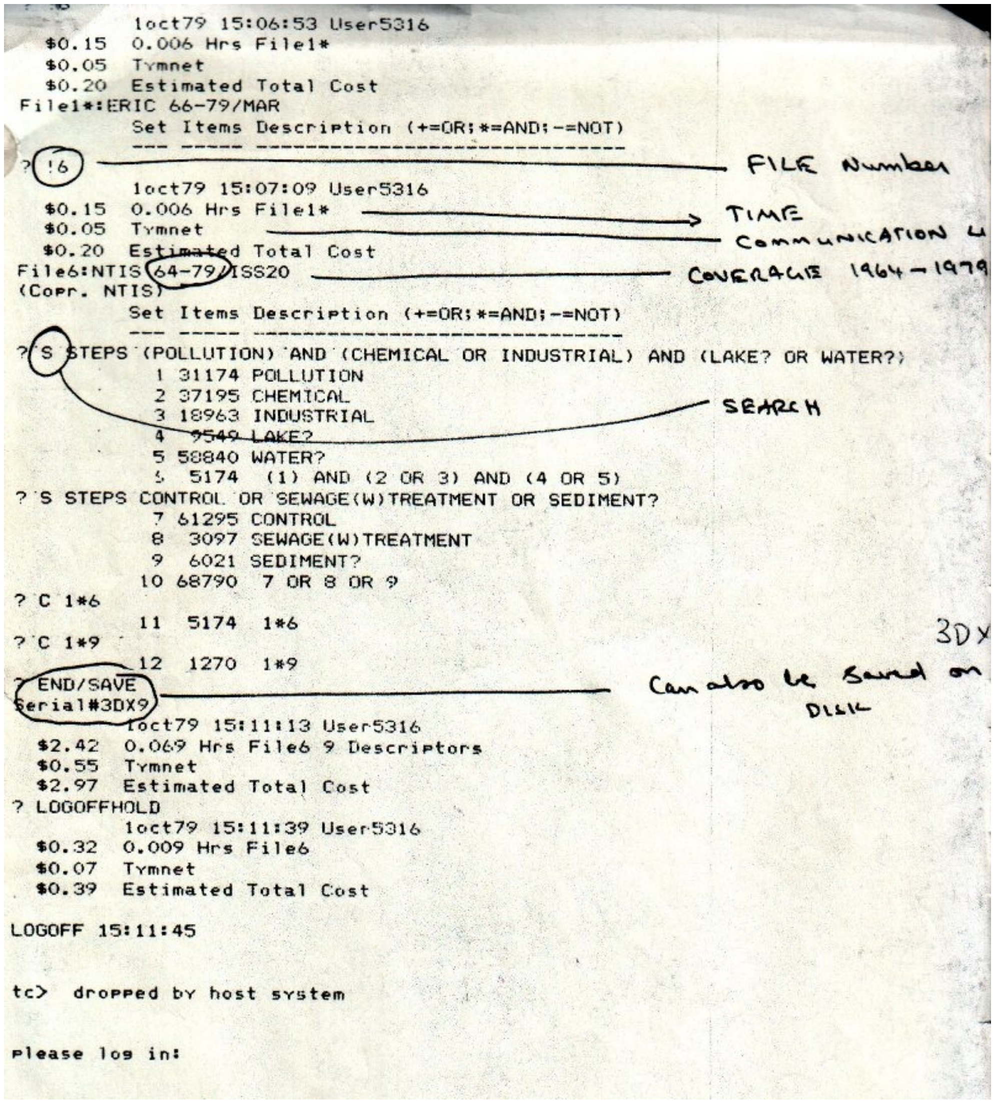
36 For example, words with more concentrated usage clustered in a small number of documents will score higher that words whose distribution is uniform across the collection of documents.

XOR (exclusive or) and *NOT*. In its simplest form, a Boolean query seeks to find those documents which contain (or not contain) terms matching the query.

Queries are constructed as logical expressions involving the query terms. Each query term may be thought of a a set of documents. When two words are *and*'*ed*, the sets are intersected; when two words are *or*'*ed*, the sets are combined (duplicate identifiers are removed). When a *NOT* is used, the *not*'*ed* set is subtracted from the first set. Parentheses are used as needed to express the order of evaluation. For example:

```
1.  COMPUTERS AND MEDICINE
2.  COMPUTERS AND (ONCOLOGY OR GASTROENTEROLOGY OR CARDIOLOGY)
3.  COMPUTERS NOT ANALOG
```

The first query would retrieve those documents containing *both* the terms *COMPUTERS* and *MEDICINE.* In the second, those documents which contained the term *COMPUTERS* and one (or more) of: *ONCOLOGY, GASTROENTEROLOGY,* or *CARDIOLOGY.* In the third example, those documents containing *COMPUTERS* but not *ANALOG.*



```
        1oct79 15:06:53 User5316
  $0.15  0.006 Hrs File1*
  $0.05  Tymnet
  $0.20  Estimated Total Cost
File1*:ERIC 66-79/MAR
        Set Items Description (+=OR;*=AND;-=NOT)
        --- ----- ------------------------------
? !6
        1oct79 15:07:09 User5316
  $0.15  0.006 Hrs File1*
  $0.05  Tymnet
  $0.20  Estimated Total Cost
File6:NTIS 64-79/ISS20
(Copr. NTIS)
        Set Items Description (+=OR;*=AND;-=NOT)
        --- ----- ------------------------------
? S STEPS (POLLUTION) AND (CHEMICAL OR INDUSTRIAL) AND (LAKE? OR WATER?)
            1 31174 POLLUTION
            2 37195 CHEMICAL
            3 18963 INDUSTRIAL
            4  9549 LAKE?
            5 58840 WATER?
            6  5174  (1) AND (2 OR 3) AND (4 OR 5)
? S STEPS CONTROL OR SEWAGE(W)TREATMENT OR SEDIMENT?
            7 61295 CONTROL
            8  3097 SEWAGE(W)TREATMENT
            9  6021 SEDIMENT?
           10 68790  7 OR 8 OR 9
? C 1*6
           11  5174  1*6
? C 1*9
           12  1270  1*9
? END/SAVE
Serial#3DX9
        1oct79 15:11:13 User5316
  $2.42  0.069 Hrs File6 9 Descriptors
  $0.55  Tymnet
  $2.97  Estimated Total Cost
? LOGOFFHOLD
        1oct79 15:11:39 User5316
  $0.32  0.009 Hrs File6
  $0.07  Tymnet
  $0.39  Estimated Total Cost

LOGOFF 15:11:45

tc>  dropped by host system

please log in:
```

FILE Number

TIME

Communication L

COVERAGE 1964-1979

SEARCH

3DX

Can also be saved on DISK

Figure 83 Example CAS Search from 1979

In Boolean based systems, the documents themselves are indexed by words or terms derived either from:

1. the documents themselves or
2. assigned to the documents from a controlled vocabulary or dictionary.

A Boolean search can be conducted in two ways:

1. Each document in the collection can be inspected and evaluated in terms of the Boolean search expression, or,
2. The Boolean search expression can be evaluated by means of an inverted index file.

In a Boolean search using an inverted file system, sets of document numbers (or other referential tokens) are intersected or joined according to the logical expression. The resulting document numbers are then used to retrieve the actual text. An example is shown in Figure 7.

For example, the query: *COMPUTERS AND MEDICINE* would be processed by intersecting the set of document numbers of those documents containing the term *COMPUTERS* with the set of document numbers containing the term *MEDICINE.* The resulting document numbers are those of documents containing both terms.

Normally, the results are presented without ranking but some systems rank the retrieved documents according to the relative frequency of query words in the document versus other documents and other techniques [Blair 1996].

Additional operators can be used such as:

1. *ADJ* requiring words to be adjacent or (*ADJ 5*) requiring the words be within 5 words of one another, or
2. *WITH* requiring the words to be in the same sentence, or
3. *SAME* requiring the words to be in the same paragraph.
4. *SYN* indicating possible term synonyms.

These are examples which were used in the IBM STAIRS system [Blair 1996] (also known as SearchManager/370 in later versions) and Lockheed's original DIALOG systems.

Wildcard truncation characters is also possible. For example *COMPUT?* would match the terms:

```
      COMPUTER
      COMPUTERS
      COMPUTED
      COMPUTATIONAL
      COMPUTING
```

Most systems of this kind retain the results of searches during a session and permit prior results to be used in new queries:

```
      1: COMPUTERS AND MEDICINE
      2: 1 AND ONCOLOGY
```

In some systems, a user might be asked to rank the importance of the terms. Documents are then scored based on the sum of user assigned weights for the search terms and only those exceeding a threshold are displayed. For example:

```
      ONCOLOGY=4
      CARDIOLOGY=5
      VIROLOGY=3
      GASTROENTEROLOGY=2
      THRESHOLD=6
      ONCOLOGY OR CARDIOLOGY OR VIROLOGY OR GASTROENTEROLOGY
```

If the threshold for document display is 6, a document with only VIROLOGY and GASTROENTEROLOGY would not be displayed (weight of 5) but another document with CARDIOLOGY and GASTROENTEROLOGY (weight of 7) would be displayed. These weights might also be used to rank the documents.

13.4.2 Simple Boolean Searching Using *grep*

The purpose of this section is to explore some simple ways to search a text database using built-in Linux utility programs such as *grep*, *egrep*, *wc*, *sort* and *uniq*.

13.4.3 Regular Expression Searching with grep

The most basic form of searching involves simply scanning the documents (the medical abstracts in this case) for words. In Linux and other systems this can be done with built-in programs such as *grep* or *egrep*. *Grep* was originally written by Ken Thompson for Unix in the early 1970s. Since then it has been extended and expanded and is now available on Linux, Unix and Windows.

At its simplest, *grep* scans one or more files for lines containing an instance of a word. For example:

```
grep -i alcohol ohsu.converted
```

will scan the file *ohsu.converted* for instances of the word *alcohol.* The lines containing the word *alcohol* will be written to standard out (stdout). *Grep* is case sensitive by default. By adding the switch *-i*, it becomes case-insensitive.

The search argument to *grep* may be a regular expression [Aho 1990]. Regular expressions are used in many utilities and programming languages (including Mumps) to specify pattern matching criteria.

For example, if you want those abstracts from *ohsu.converted* that contained any word beginning with the term *alcohol*, you could type:

```
grep -i "alcohol[a-z]*" ohsu.converted
```

where the figure *[a-z]* means any letter in the range of *a* through *z* and the asterisk means repeated zero or more times.

Similarly, specific combinations of letters may be used such as:

```
grep -i "alcohol[ic|ics]" ohsu.converted
```

which means that the word matched must begin with *alcohol* and be followed by either *ic* or *ics* (*alcoholic* or *alcoholics*).

To see the count of number of documents containing, for example, the words *alcoholic* or *alcoholics* you type:

```
grep -i "alcohol[ic|ics]" ohsu.converted | wc -l
```

which passes (*pipes*) the output from *grep* to *wc* which, in turn, displays the number of lines it received. Since each document in this file is on one (long) line, the number written by *wc* is the number of documents containing one of the words. Note: all the *stdout* output of *grep* is passed to *wc* so you will only see the final line count. The *-l* parameter to *wc* tells it to output only the line count.

The Perl language also has an extended pattern match features and is very well suited for searches such as these.

13.4.4 Boolean Searching with Regular Expressions

Note: in the experimental files supplied and built, by default, are *ohsu.medline* and *ohsu.converted* and these are the names used in this text. However, in the *bash* script files, the *ohsu* prefix is a *bash* variable substitution.

The *grep* searches shown above can be extended to handle Boolean expression based searches.

For example, if the query is to find those abstracts containing:

```
alcohol* AND gambl*
```

you could construct the following command line[37]:

```
grep "alcohol[a-z]*" ohsu.converted | \
  grep "gambl[a-z]*" ohsu.converted | wc -l
```

which first finds all articles containing words that begin with *alcohol* and then, from this set, selects those articles containing words that begin with *gambl*. This effectively *ands* the result which is piped to *wc* for the final count.

Alternatively, the expression:

```
alcohol OR gambling
```

could be rendered as shown in Program 45[38]

37 The backslash at the line end indicates that the line is continued on the next line.

38 Note that the prefix for the file with the *.converted* suffix is obtained from the file DBPREFIX. DBPREFIX is created by *index.script* which should be executed prior to doing these examples. It stores the file name prefix (*ohsu* by default) in DBPREFIX and the number of documents processed in the file MAXDOCS. The file DocCount is also created by *index.script* and contains the number of documents actually in the indexed collection. This number may be significantly less that MAXDOCS as some documents may be eliminated during indexing.

```
#!/bin/bash
# Copyright 2014 Kevin C. O'Kane
# boolean-grep-01.script

if [ -f "DBPREFIX" ]
        then
    db=`cat DBPREFIX`
        echo "Using $db.converted"
else
    echo "File DBPREFIX not found"
    exit
         fi

pid=$$
grep "alcohol[a-z]*" $db.converted > /tmp/$pid.tmp
grep "gambl[a-z]*" $db.converted  >> /tmp/$pid.tmp
sort /tmp/$pid.tmp | uniq | wc -l ; rm /tmp/$pid.tmp
```

Program 45 Boolean OR search with grep

The *$$* figure in Program 45 is a builtin *bash* variable that contains the process ID of the current shell. Since no two process IDs are concurrently same[39], this value will be unique. The process ID is stored in the *bash* shell variable *pid*.

The first *grep* command stores its results in a temporary file placed in */tmp* whose name consists of the process ID followed by *.tmp*.

The second *grep* concatenates[40] its results onto the end of this file.

The last line runs several programs with the output of each becoming the input of the next.

The *sort* command sorts the results from *$pid.tmp* based on the first whitespace delimited token in each line. In our case, this is the unique (and identifying) file offset of the abstract in the original OHSUMED file.

The output of the *sort* is then piped to *uniq* which collapses duplicate lines to one line. For example, a document which contains both word stems would be in the output file twice.

39 Process IDs are ultimately re-used but the same process ID is never assigned to two processes executing at the same time.

40 the operator >> means concatenate output at the end of the target file.

The output of *uniq* is piped to *wc* which counts the number of abstracts that contain either or both of the search terms. The *rm* command deletes the temporary file.

A more complex query such as:

```
(alcohol OR gambling) AND smoking
```

could be written as shown in Program 46. In Program 46 the result of the OR operation from lines 6 and 7 in */tmp/$pid.2.tmp,* becomes input to the AND operation in the third *grep* command.

```
1  #!/bin/bash
2  # Copyright 2104 Kevin C. O'Kane
3  # boolean-grep-02.script
4
5  if [ -f "DBPREFIX" ]
6         then
7         db=`cat DBPREFIX`
8         echo "Using $db.converted"
9  else
10         echo "File DBPREFIX not found"
11         exit
12         fi
13
14 pid=$$
15 grep "alcohol[a-z]*" $db.converted > /tmp/$pid.1.tmp
16 grep "gambl[a-z]*" $db.converted  >> /tmp/$pid.1.tmp
17 sort /tmp/$pid.1.tmp | uniq > /tmp/$pid.2.tmp
18 grep "smoking" /tmp/$pid.2.tmp | wc -l; rm /tmp/$pid.*.tmp
```

Program 46 Boolean AND and OR search with grep

An alternative *grep* expression for line 6 Program 46 might be:

```
grep "alcohol[a-z]*\|gambl[a-z]*" ohsu.converted > /tmp/$pid.1.tmp
```

where the OR operation is performed by *grep* (the \| means OR). This would eliminate the second *grep* on line 7. However, in tests[41], this modification actually resulted in slower execution[42]. Not all intuitive solutions work! Always test.

Program 46 could be modified to improve performance by piping the output of line 8 directly as input to the *grep* in line 9. This would reduce the number of intermediate files.

Another variation of Program 46 would involve parallel execution of the *grep* functions on lines 6 and 7. On a multi-core processor, this should improve execution substantially. The modified code is given in Program 47. In this program, the *grep* functions on lines 6 and 7 are executed in parallel (because of the & at the end of the lines). The two output files become input to the *sort* function on line 9. The *bash wait*[43] command on line 8 causes the script to pause until all background processes in the current shell have completed.

41 See *boolean-grep-02b.script*

42 On a 6 core AMD processor under Linux Mint LMDE in April, 2014. Your mileage may vary.

43 Without arguments, *wait* waits for all background processes to complete. If you specify one or more process IDs as arguments to *wait*, *wait* waits only for those you specify. The process ID of the most recent command is in the *bash* builtin variable $! which can be assigned to a shell variable for later reference.

```
#!/bin/bash
# Copyright 2104 Kevin C. O'Kane
# boolean-grep-02a.script

if [ -f "DBPREFIX" ]
       then
       db=`cat DBPREFIX`
       echo "Using $db.converted"
else
        echo "File DBPREFIX not found"
        exit
        fi

pid=$$
grep "alcohol[a-z]*" $db.converted > /tmp/$pid.1.tmp  &
grep "gambl[a-z]*" $db.converted  > /tmp/$pid.2.tmp  &
wait
sort /tmp/$pid.1.tmp /tmp/$pid.2.tmp | uniq > /tmp/$pid.3.tmp
grep "smoking" /tmp/$pid.3.tmp | wc -l; rm /tmp/$pid.*.tmp
```

Program 47 Parallel Boolean search with grep

Rather than just print the count of the number of documents fulfilling the search criteria, we may want to know the specific document numbers and file offsets. This can be done using a command sequence such as shown in Program 48

Final output (from *egrep*) will appear on stdout. *Egrep* is an extended version of *grep* that permits additional operators, the + operator in this case. In the above, the *egrep* receives the output of the final *grep* and matches the first two numeric sequences on each line. The ^ operator means *at the beginning* of line and the + means 1 or more instances of the preceding token (a number in the range 0 through 9 in this case). A blank must separate the two numbers. The output, determined by the *-o* switch, will only consist of the portion of the input line matched (that is, the first two numbers)[44].

44 The ^ operator means beginning of line, the $ operator means end of line.

```
#!/bin/bash
# Copyright 2014 Kevin C. O'Kane
# boolean-grep-03.script

if [ -f "DBPREFIX" ]
       then
       db=`cat DBPREFIX`
       echo "Using $db.converted"
else
        echo "File DBPREFIX not found"
        exit
        fi

pid=$$
grep "alcohol[a-z]*" $db.converted > /tmp/$pid.1.tmp
grep "gambl[a-z]*" $db.converted  >> /tmp/$pid.1.tmp
sort < /tmp/$pid.1.tmp | uniq > /tmp/$pid.2.tmp
grep "smoking" /tmp/$pid.2.tmp | egrep -o "^[0-9]+ [0-9]+ "
rm /tmp/$pid.*.tmp
```

Program 48 Display document numbers from grep search

To display the titles from the above, use Programs 49 and 50.

```
#!/bin/bash
# Copyright 2014 Kevin C. O'Kane
# boolean-grep-04.script

if [ -f "DBPREFIX" ]
       then
       db=`cat DBPREFIX`
       echo "Using $db.converted"
else
        echo "File DBPREFIX not found"
        exit
        fi

pid=$$

grep "alcohol[a-z]*" $db.converted > /tmp/$pid.1.tmp
grep "gambl[a-z]*" $db.converted  >> /tmp/$pid.1.tmp
sort < /tmp/$pid.1.tmp | uniq > /tmp/$pid.2.tmp
grep "smoking" /tmp/$pid.2.tmp | egrep -o "^[0-9]+ [0-9]+ " | get-docs.mps
rm /tmp/$pid.*.tmp
```

Program 49 Display titles from grep search

where *get-docs.mps* is given in Program 50[45].

45 Note: The indexing procedure in *index.script* creates the global array ^*title(d)* that contains the titles for the given document number (*d*) which we assume exists for this program.

```
#!/usr/bin/mumpsRO
# boolean.mps Feb 14, 2014
 # Copyright 2014 Kevin C. O'Kane

    set i=0

    for  do
    . read line
    . if '$test break
    . set docnbr=$p(line," ",2)
    . if '$data(^title(docnbr)) quit
    . write docnbr,?10,$e(^title(docnbr),1,80),!

    write "Only titles in ^title() displayed.",!
    halt
```

Program 50 Program to retrieve and display titles

As an example, the query corresponding to *(alcohol OR gambling) AND smoking* yields the results in Figure 84. based on the first 5,000 abstracts (long lines truncated).

```
1461     Health and economic implications of a tobacco-free society.
1749     Hip fracture and the use of estrogens in postmenopausal women. The Framingham
3012     Pulmonary impairment in a cotton textile factory in Nigeria: is lifetime alcohol
9        Alcohol and the elderly: relationships to illness and smoking.
```

Figure 84 Example Boolean search with grep/egrep

When multiple functions are run on the same line and if the output of one becomes the input of the next, on a machine with multiple processors, the system will schedule normally the separate functions concurrently onto different processors thus improving speed by parallel execution. *Bash* also has facilities to permit programs initiated by multiple command lines to be executed concurrently. However, even with parallel execution, the operations described above are sequential in nature and inherently slow on very large collections.

Finally, it should be noted that functions written in most programming languages have the ability to create and issue shell commands. In GPL Mumps this can be done several ways including the *shell* command and the

$zsystem() built-in function. The *shell* command can be used to read or write (but not both at the same time) from/to the shell. In the C/C++ language, the *system()* and *popen()* functions provide this functionality.

Overall speed of these operations can be increased on a multi-core machine if the underlying database is *sharded (t*hat is, split into multiple pieces) and the search conducted in parallel on each shard or piece with the final results combined (similar to Map-Reduce).

As an exercise, write a program that accepts a Boolean expression involving vocabulary terms and the operators AND, OR, and NOT, parses the expression, and then executes the appropriate *bash* shell commands to search the database.

13.4.5 A Boolean Search in Mumps

The code in Program 51 is a simple full-text, sequential, boolean search of the OHSUMED collection written in Mumps. Program 51 produces output as shown in Figure 85. It operates on the modified OHSUMED database *ohsu.converted* an example of which was shown in Figure Error: Reference source not found on page Error: Reference source not found.

The program first reads a query and then loads it into a Mumps internal line buffer. The query consists of one or more search terms separated by the operators *AND (&), OR (!), NOT (~),* and matching sets of parentheses such as:

```
(term1 & term 2) | (term3 & term4) & ~ term5
```

```
(term1 AND term 2) OR (term3 AND term4) AND NOT term5
```

where *term1,...term5* are words or terms from the vocabulary.

The loop extracts tokens (*$zwparse*) from the query buffer and builds a Mumps expression in the string variable *exp*. This expression is a Mumps translation of the user's Boolean query using Mumps logical operators and the Mumps *$find()* function. So, for example, a user input expression such as:

```
(apples AND oranges) OR pears
```

internally becomes:

```
($find(line,"apples")&$find(line,"oranges"))!$find(line,"pears")
```

Once a user query is converted to a Mumps expression, the syntax of the expression is tested. The expression is then applied to consecutive input lines from *ohsu.converted* in a loop.

Each line from *ohsu.converted* read into the variable *line* contains the full text of an original abstract, converted to lower case, stemmed, and devoid of punctuation.

The loop executes the expression in *exp* in an *if* statement. If the expression results in *true*, the program extracts from *line* the offset of the document in the original file and the document number. Based on the document number, the title[46] is displayed along with the document number. The abstract could also be displayed by using the offset of the original document in the original file.

46 We assume the ^*title(docnbr)* has been previously built and contains the text of the titles for all document numbers *docnbr*.

```
#!/usr/bin/mumpsRO
# Copyright 2014 Kevin C. O'Kane
# boolean.mps February 14, 2014
# assumes that ^titles(docnbr) exists

        read "Enter query terms ",query

        set query=$zlower(query)
        set i=$zwstore(query)

        set exp=""

        for w=$zwparse do
        . if w="" break
        . if $find("()",w) set exp=exp_w continue
        . if w="|"!(w="OR") set exp=exp_"!" continue
        . if w="~"!(w="NOT") set exp=exp_"'" continue
        . if w="&"!(w="AND") set exp=exp_"&" continue
        . set exp=exp_"$f(line,"""_w_""")"

        write !,"Mumps expression to be evaluated on the data set: ",exp,!!

        set $noerr=1  // turns off error messages
        set line="  " set i=@exp  // test trial of the expression
        if $noerr<0 write "Expression error number ",-$noerror,!  got to again

        open 2:"MAXDOCS,old"
        if '$test write "MAXDOCS not found.",! halt
        use 2 read M close 2
        use 5

        open 2:"DBPREFIX,old"
        if '$test write "DBPREFIX not found.",! halt
        use 2 read P close 2
        use 5

        set file=P_".converted,old"

        open 1:file
        if '$test write "file error",! halt

        set i=0

        for j=1:1:M do
        . use 1
        . read line
        . if '$test break
        . if @exp do
        .. set off=$piece(line," ",1)
        .. set docnbr=$piece(line," ",2)
        .. use 5
        .. write docnbr,?10,$e(^title(docnbr),1,80),!

        use 5
        write !,M," documents searched",!!
        halt
```

Program 51 Boolean search in Mumps

The program in Program 51 is slow, however, because it searches each document sequentially. A better way would be to build an inverted index of all significant words (with file offset pointers into the original OHSUMED file) and process the queries against the inverted index. This is discussed below.

```
Enter query: drink & alcohol

Mumps expression to be evaluated on the data set: $f(line,"drink")&$f(line,"alcohol")

4        Drinkwatchers--description of subjects and evaluation of laboratory markers of
7        Bias in a survey of drinking habits.
1490     Self-report validity issues.
1491     A comparison of black and white women entering alcoholism treatment.
1492     Predictors of attrition from an outpatient alcoholism treatment program for
1493     Effect of a change in drinking pattern on the cognitive function of female social
1494     Alcoholic beverage preference as a public statement: self-concept and social image
1496     Influence of tryptophan availability on selection of alcohol and water by men.
1497     Alcohol-related problems of children of heavy-drinking parents.
1499     Extroversion, anxiety and the perceived effects of alcohol.
2024     Psychiatric disorder in medical in-patients.

3648 documents searched

-----

Enter query: (drink | alcohol) & problem

Enter query terms
Mumps expression to be evaluated on the data set: ($f(line,"drink")!$f(line,"alcohol"))&$f(line,"problem")

7        Bias in a survey of drinking habits.
1056     Reduction of adverse drug reactions by computerized drug interaction scree
1069     Suicide attempts in antisocial alcoholics.
1487     Childhood problem behavior and neuropsychological functioning in persons a
1496     Influence of tryptophan availability on selection of alcohol and water by
1497     Alcohol-related problems of children of heavy-drinking parents.
1959     Native American postneonatal mortality.
2024     Psychiatric disorder in medical in-patients.
4430     Family history of problem drinking among young male social drinkers: behav
4435     Dose-related effects of alcohol among male alcoholics, problem drinkers an
4439     Reliability and validity of the MAST, Mortimer-Filkins Questionnaire and C
4440     Social drinking and cognitive functioning in college students: a replicati
5453     Fatal occupational injuries of women, Texas 1975-84.
7532     Substance use and mental health problems among sons of alcoholics and cont
...
```

Figure 85 Boolean search results from Program 51

13.5 Genomic Retrieval

Another text based example is in the area of Bioinformatics. Here, researchers with DNA or protein sequences need to search massive databases for similar and, sometimes, only distantly related, DNA or protein sequences.

For example, consider the the DNA sequence[47] shown in Figure 86.

```
>gi|2695846|emb|Y13255.1|ABY13255 Acipenser baeri mRNA for immunoglobulin heavy chain,
TGGTTACAACACTTTCTTCTTTCAATAACCACAATACTGCAGTACAATGGGGATTTTAACAGCTCTCTGTATAATAATGA
CAGCTCTATCAAGTGTCCGGTCTGATGTAGTGTTGACTGAGTCCGGACCAGCAGTTATAAAGCCTGGAGAGTCCCATAAA
CTGTCCTGTAAAGCCTCTGGATTCACATTCAGCAGCGCCTACATGAGCTGGGTTCGACAAGCTCCTGGAAAGGGTCTGGA
ATGGGTGGCTTATATTTACTCAGGTGGTAGTAGTACATACTATGCCCAGTCTGTCCAGGGAAGATTCGCCATCTCCAGAG
ACGATTCCAACAGCATGCTGTATTTACAAATGAACAGCCTGAAGACTGAAGACACTGCCGTGTATTACTGTGCTCGGGGC
GGGCTGGGGTGGTCCCTTGACTACTGGGGGAAAGGCACAATGATCACCGTAACTTCTGCTACGCCATCACCACCGACAGT
GTTTCCGCTTATGGAGTCATGTTGTTTGAGCGATATCTCGGGTCCTGTTGCTACGGGCTGCTTAGCAACCGGATTCTGCC
TACCCCCGCGACCTTCTCGTGGACTGATCAATCTGGAAAAGCTTTT
```

Figure 86 Example DNA Sequence

The sequence in Figure 86 is that for a known and archived DNA sequence. It is shown in what is referred to as FASTA format[48]. In FASTA format, the first line gives the name and library accession numbers of the sequence. The subsequent lines are the DNA nucleotide codes.

All DNA is expressed as a sequence of letters from a four character alphabet. The letters are A, C, G, and T which represent the chemicals Adenine, Cytosine, Guanine, and Thymine, respectively[49].

For many years several large national and international agencies have been collecting DNA sequences that have been identified by researchers. There are millions of such sequences and the number grows daily.

Researchers, upon identifying a new sequence, need to find out if the sequence is already known and what it may be related to whether known or unknown. But DNA sequences present a problem. Even if they are for the same genetic function in the same species, they may vary slightly from on individual to another. For example,

47 Note: there are also databases of protein sequences not discussed here. See the web page for the National Center for Biotechnology Information (NCBI) for additional details:
http://www.ncbi.nlm.nih.gov/

48 http://en.wikipedia.org/wiki/FASTA_format

49 Note: protein sequences are ordinarily written with an alphabet of 20 letters corresponding to the amino acids found in proteins.

some people have red hair, others have blond. This is determined by DNA sequences that are similar but differ slightly. Similarly, a DNA sequence in one species is likely to be related to a similar sequence in another species. Due to evolution, these will vary and, where the evolutionary distance is great, the variance may be large.

The question is, how do you build a search engine that can quickly look for similar but not identical sequences (called *homologues*)? In some cases, the relationship may be very faint but nonetheless real.

There are several IR systems to do these searches but a program known as BLAST[50] (Basic Local Alignment Sequencing Tool) is one of the most popular. It can be used to find similar sequences in online databases of known DNA and protein sequences.

If you submit the sequence from Figure 86 to NCBI BLAST (National Center for Biotechnology Information), they will conduct a search of their *nr* database. The result will be a ranked list of hits of sequences in the database ordered according to their similarity to the query sequence. BLAST will also give a statistical evaluation of whether the match is significant and not due to random chance alone. Sequences found whose similarity scores exceed a threshold are displayed.

When the sequence from Figure 86 was submitted to BLAST, a number a matches were returned. One of these is shown in Figure 87. In Figure 87 BLAST displays the sections from the query sequence that match a portion of the sequence from the database. The numbers at the beginning and ends of the lines are the starting and ending points of the subsequence (relative to one, the start of all sequences). Where there are vertical lines between the query and the subject, there is an exact match. Where there are blanks, there was a mismatch.

It should be clear that, even though the subject differs from the query in many places, the two have a high degree of similarity. BLAST calculates a score and an expectation value which is *“...the number of hits one can* "*expect*" *to see by chance when searching a database of a particular size. It decreases exponentially as the Score (S) of the match increases*[51] *...*“ In this case, the likelihood is 4e-33 (4 x 10^{33}) that these two sequences are related by chance alone.

However, note the first lines from the found sequence and the query sequence. They are from different organisms but nonetheless related.

50 http://blast.ncbi.nlm.nih.gov/Blast.cgi

51 http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs

```
>gb|U17058.1|LOU17058 Lepisosteus osseus Ig heavy chain V region mRNA, partial cds

 Score =  151 bits (76),  Expect = 4e-33
 Identities = 133/152 (87%), Gaps = 0/152 (0%)
 Strand=Plus/Plus

Query  242  TGGGTGGCTTATATTTACTCAGGTGGTAGTAGTACATACTATGCCCAGTCTGTCCAGGGA  301
            |||||||| ||||||||| | | ||| || | |||||||||| |||||||||||||||||
Sbjct  4    TGGGTGGCGTATATTTACACCGATGGGAGCAATACATACTATTCCCAGTCTGTCCAGGGA  63

Query  302  AGATTCGCCATCTCCAGAGACGATTCCAACAGCATGCTGTATTTACAAATGAACAGCCTG  361
            |||||| |||||||||||||| ||||||| |    |||||| ||||| |||| |||||||
Sbjct  64   AGATTCACCATCTCCAGAGACAATTCCAAGAATCAGCTGTACTTACAGATGAGCAGCCTG  123

Query  362  AAGACTGAAGACACTGCCGTGTATTACTGTGC  393
            ||||||||||||||||| ||||||||||||||
Sbjct  124  AAGACTGAAGACACTGCTGTGTATTACTGTGC  155
```

Figure 87 Example BLAST Result

14 Alternative Vocabulary Encodings

Documents and queries may be re-encoded in other ways which may also be used for indexing. The following describe a few methods that have been used over the years.

14.1 Word Truncation

Salton's early work was done on IBM 7090 series computers. These were six-bit, character based machines with a word size of 36 bits (6 six-bit characters per word).

When using 6 bit characters, there are only 64 possible character codes available. Consequently, these machines normally only handled upper case letters[52].

The 7090 series machines, as was the case with most machines of that era, had word addressable memories. That is, the smallest unit of storage that could be addressed was a 36 bit word. This simplified the system addressing circuitry considerably by reducing the number of address lines that needed to be implemented and calculated. Also, these early machines had relatively small (by today's standards) expensive magnetic core memory.

52 It wasn't until the mid-60s and the introduction of the IBM 360 series that the eight bit byte became widely available.

To save space and increase program efficiency, many of Salton's early experiments were conducted using only the first six characters from each word in a document. In practice, however, this worked relatively well in that much of the information content of English words is in the first few syllables.

14.2 Soundex Encoding

Soundex is a technique (patent number 1,261,167 on April 2, 1918) to convert words that sound like one another into common codes. It was originally (and still is) used for telephone directory assistance to permit operators to quickly access phone numbers based on the sound of a name rather than on a detailed spelling of the name.

In Soundex, the operator manually generates a alphnumeric code based on what the name sounded like and uses this to locate a page in the directory likely to contain the name. Then operator asks the caller the first name and then confirms the identity with the address ('*do you mean Jane Smith on Willow Street?*').

Mumps has a builtin function *$zzSoundex()* which returns a Soundex code for a string passed as its argument. Unlike traditional Soundex, the code can be longer or shorter than four characters, depending on the size of the string passed. This returned code, can, of course, be truncated or padded with zeros as needed.

It is possible to re-code a document collection to fixed length Soundex codes and base all the indexing on 'words' consisting of Soundex codes. The results can be very good.

Program 52 gives an example of converting the dictionary in *^dict* to Soundex codes a sample of which are shown in Figure 88.

```
#!/usr/bin/mumps
# ~/Medline2012/stems.mps Nov 11, 2014
# Copyright 2014 Kevin C. O'Kane
# convert data base to word stems

        for w=$order(^dict(w)) do
        . set sndx=$zzsoundex(w)
        . set ^sndx(sndx,w)=""

        for w=$order(^sndx(w)) do
        . write w,": "
        . for x=$order(^sndx(w,x)) write " ",x
        . write !!
```

Program 52 *soundex.mps*

```
g42:   gels glas glass gliosis golgi

g421:  glossopharyngeal glycopeptide glycophorin glycoprotein glycopyrrolate

g422:  glasgow glucagon glucagonoma glucokinase glucosamine glucose glucose-induce
       glucose-infus glucose-stimulate glycogen glycogenolysis glycosaminoglycan glycosuria
       glycosylase glycosylate

g423:  galactose galactosemia galactosylate gallstone glycation

g424:  glycocalicin glycocalyx glycol glycolipid glycolysis glycolytic

g425:  glaucoma glaucomat glcnac gleason glucan gluconate gluconeogenesis gluconeogenic
       glycaemic glycan glycemia glycemic glycine glycine-extend glycoconjugate

g426:  glucocerebrosidase glucocorticoid glucocorticoid-induce glucocorticosteroid
       glucuronidate glucuronide glycerol

g43:  glott gold golyte
```

Figure 88 Soundex codes

14.3 N-gram Encoding

During World War II, *n-grams* were developed by cryptographers to break substitution ciphers [Salton 1983]. An n-gram is fixed length strings of *n* characters from the text (numbers, punctuation and blanks omitted).

Applying n-grams to indexing, the text, stripped of non-alphabetic characters, is treated as a continuous stream of data that is segmented into non-overlapping fixed length words. These words can then form the basis of the indexing vocabulary often with good results. Queries terms are converted to n-grams consisting of all possible overlapping strings of length *n*.

15 Evaluation of Retrieval Systems

In order to compare the many approaches and algorithms for information retrieval, we need a way to evaluate the results. Two of the more widely used metrics are discussed next: *precision* and *recall*.

In many experimental systems a set of trial queries is is developed. Then experienced researchers inspect the collection of documents and develop for each query a list of documents they feel most correctly answer the query. The results from the computer system for each query are then compared with these and scores are developed. These scores are presented as precision/recall graphs.

15.1 Precision and Recall

Two important metrics of information storage and retrieval system performance are *precision* and *recall*. *Precision* measures the degree to which the documents retrieved are relevant and *recall* measures the degree to which the system can retrieve all relevant documents.

For example, if a system responds to a query by retrieving 10 documents from the collection and of these, 8 are relevant and 2 are irrelevant and if the collection actually has 16 relevant documents, we say that the recall is 50% and the precision is 80%. That is, only 50% of the relevant documents were recalled but of those presented, 80% were correct.

For example, suppose there were 10 relevant documents in the collection and the top ten ranked results of a query are shown in Figure 89.

Rank	Relevant?	Recall	Precision
1	yes	0.1	1.0
2	yes	0.2	1.0
3	no	0.2	0.67
4	yes	0.3	0.75
5	yes	0.4	0.80
6	no	0.4	0.67
7	no	0.4	0.57
8	yes	0.5	0.63
9	no	0.5	0.56
10	yes	0.6	0.60

Figure 89 Precision/recall example

In general, as recall increases, precision declines. For example, in the query mentioned in the previous paragraph, if by setting thresholds lower the system responds with 20 documents instead of 10 and if 12 of these are relevant but 8 are not, the recall has increased to 75% but the precision has fallen to 60%.

In most systems, as you lower thresholds and more documents are retrieved, the recall will rise but the precision will decline. In an ideal system, however, as thresholds are lowered, recall increases but precision remains 100%.

Indexing terms effect the precision/recall results. Generally speaking, terms of low frequency tend to increase the precision of a system's responses at the expense of recall as these tend to reference more narrowly defined concepts. On the other hand, terms of high frequency tend to increase recall at the expense of precision as these are often more broadly defined. Identifying those terms which strike a balance is a major goal of any system.

Salton [Salton, 1971] used precision-recall graphs similar to the one shown in Figure 90 in order to compare the results of different retrieval experiments. Those experiments which resulted in a slower drop off in precision as recall increases represent improvement in technique.

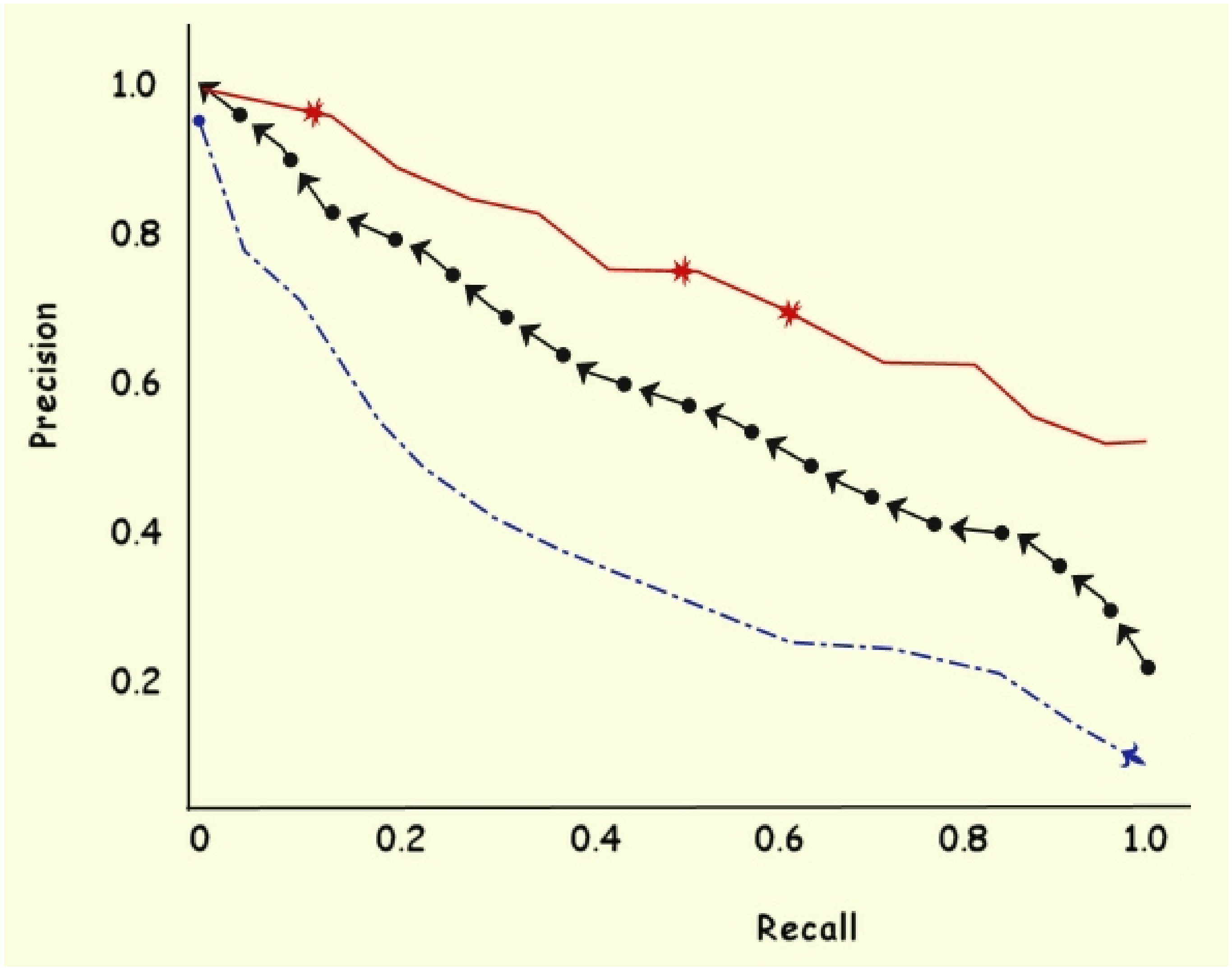


Figure 90 Precision/recall graph

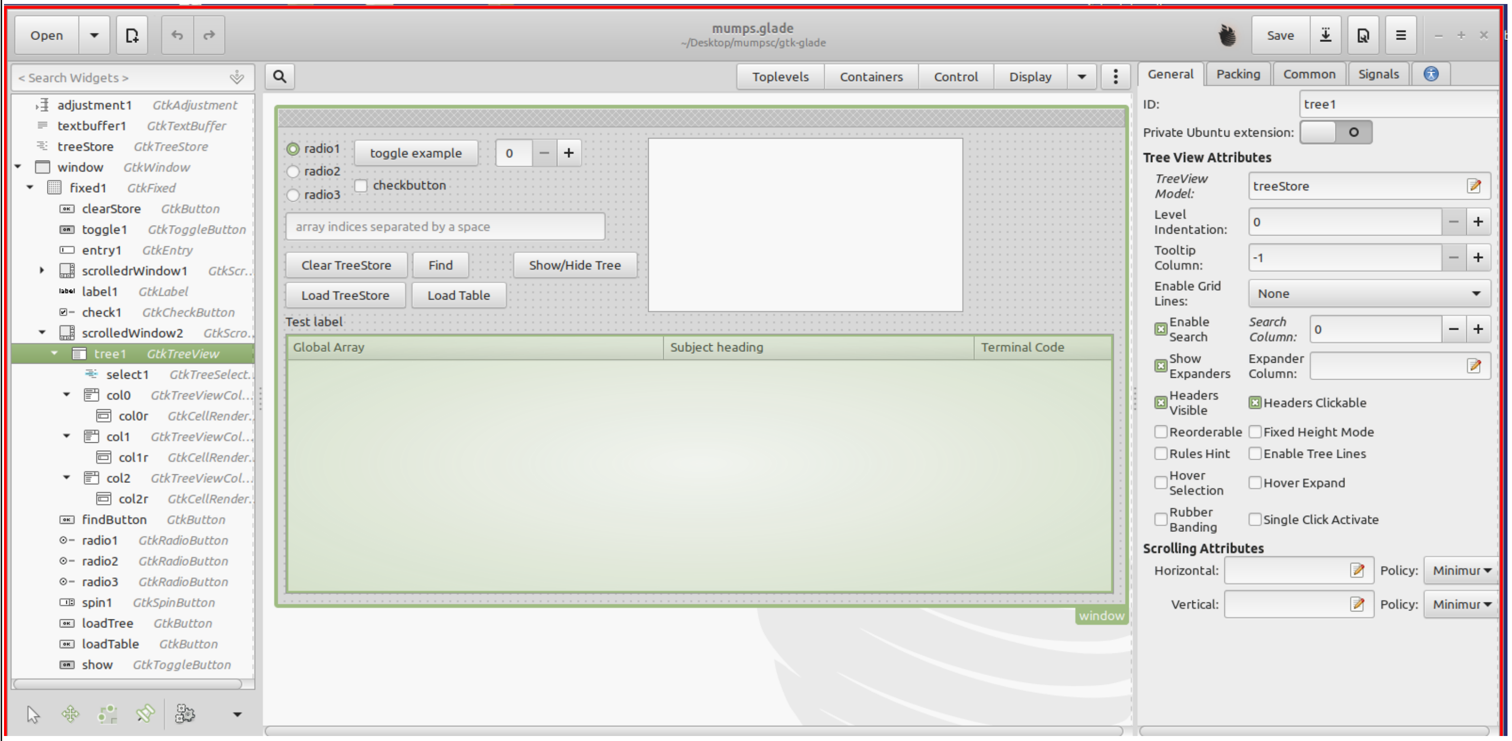
16 Building GUI Retrieval Apps

The Mumps compiler and interpreter have a facility to assist in building desktop GUI applications in conjunction with Glade and GTK. If your system is compatible with these packages, you should be able to quickly develop desktop apps to visualize you retrieval system.

The systems works as follows:

1. Use Glade to design your desktop application window. Glade is a drag and drop design tool to be used with GTK. For each functional widget in you app (button, slider, text view box, tree view box, etc.), you will name the widget and identify which actions it will respond to. For example, if you app has a button in it, you will probably tell Glade that the button should emit a *clicked* signal if the user mouse clicks on the button.
2. Glade generates an XML file describing your application. The Mumps language processor has a program, written in Mumps that will parse the XML looking for widgets. For each widget, the program will generate code that will be used to display and process the widgets and the signals they emit.
3. There is a template program named *gtk.mps* that, when compiled with the Mumps compiler, will include the files built in the step above and build a functional GTK GUI.
4. For each signal you established for each widget, you the executable generated by compilation of *gtk.mps* will call a Mumps program where you will instert code to do what you want the signal to do. For example, search for a keyword. The Mumps interpreter and compiler have simplified versions of the GTK functions that allow you to manipulate the presentation in the app.

The open source program *Glade* allows the user to design the layout of a desktop GUI app by dragging and dropping GUI widgets (buttons, text boxes, etc.) onto a canvas. Figure 91 gives an example that shows several typical widget types in use.



```
adjustment1      GtkAdjustment
textbuffer1      GtkTextBuffer
treeStore        GtkTreeStore
window           GtkWindow
  fixed1         GtkFixed
    clearStore   GtkButton
    toggle1      GtkToggleButton
    entry1       GtkEntry
    scrolledrWindow1  GtkScr...
    label1       GtkLabel
    check1       GtkCheckButton
    scrolledWindow2   GtkScro...
      tree1      GtkTreeView
        select1  GtkTreeSelect...
        col0     GtkTreeViewCol...
          col0r  GtkCellRender...
        col1     GtkTreeViewCol...
          col1r  GtkCellRender...
        col2     GtkTreeViewCol...
          col2r  GtkCellRender...
    findButton   GtkButton
    radio1       GtkRadioButton
    radio2       GtkRadioButton
    radio3       GtkRadioButton
    spin1        GtkSpinButton
    loadTree     GtkButton
    loadTable    GtkButton
    show         GtkToggleButton
```

ID: tree1
Private Ubuntu extension:

Tree View Attributes

TreeView Model: treeStore
Level Indentation: 0
Tooltip Column: -1
Enable Grid Lines: None
[✓] Enable Search — Search Column: 0
[✓] Show Expanders — Expander Column:
[✓] Headers Visible — [✓] Headers Clickable
[ ] Reorderable — [ ] Fixed Height Mode
[ ] Rules Hint — [ ] Enable Tree Lines
[ ] Hover Selection — [ ] Hover Expand
[ ] Rubber Banding — [ ] Single Click Activate

Scrolling Attributes

Horizontal: — Policy: Minimum
Vertical: — Policy: Minimum

Figure 91 Glade Canvas

When you save a Glade canvas it appears in your directory as a file with the *.glade* extension. The contents of this file is in XML giving the details on your design.

Included with the Mumps distribution in the directory *gtk-glade* is a script file named *appBuild,script* and a Mumps program named *extractWidgets.mps*. The script file:

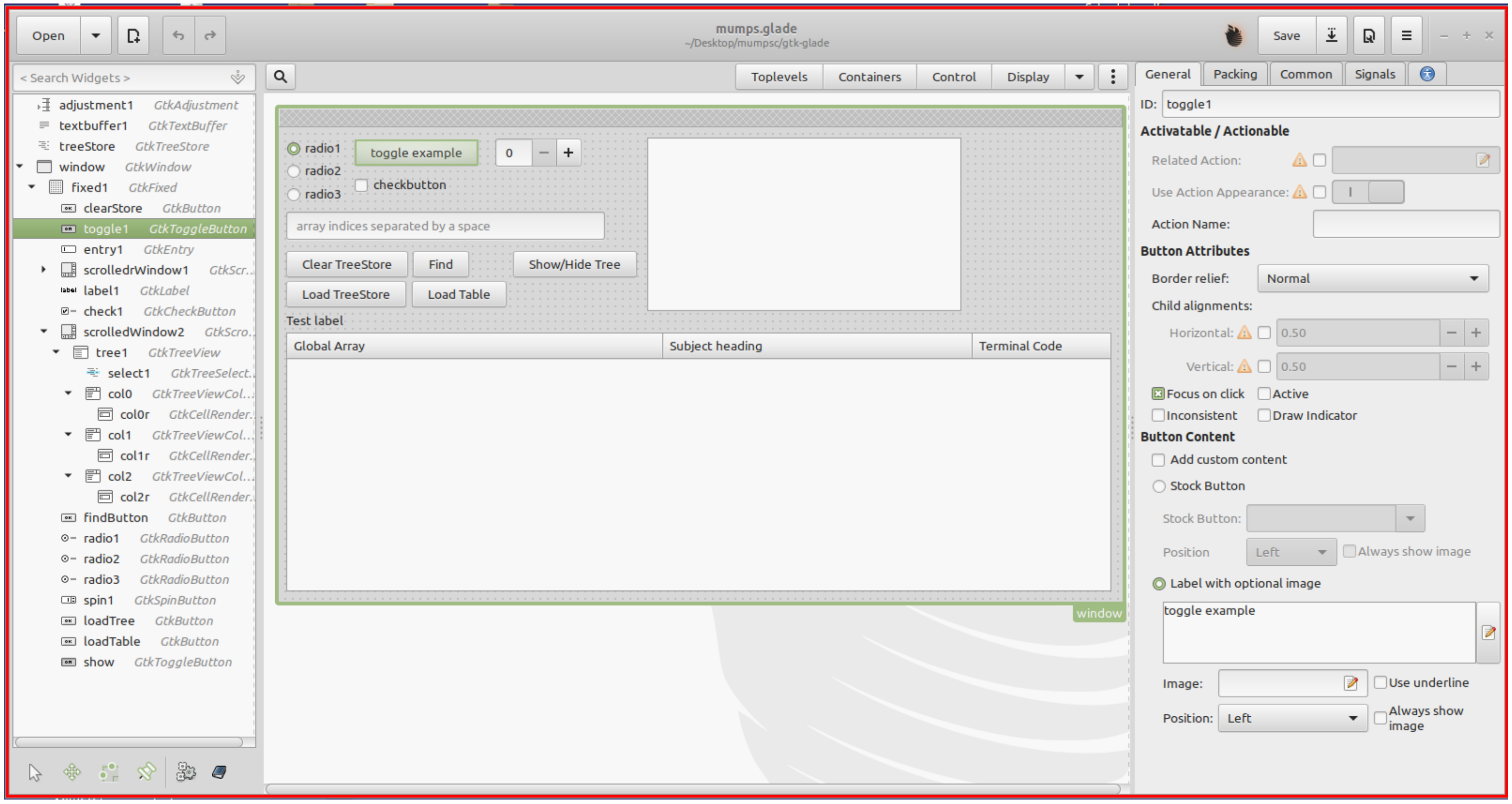
1. runs the Mumps file which reads the file *.glade* file from above and builds several files;
2. compiles (using the Mumps compiler) the file *gtk.mps* which includes the files from the previous step and creates an executable file named *gtk* which will render your GUI application on the screen.

Among the files created by *extractWidgets.mps* are several files containing Mumps programs to service the actions to be performed by interacting with the on-screen GUI. There will be one file for each signal defined for each widget. The files will have names of the form:

```
    on.widgetName.clicked.mps
```

where *widgetName* is the name of the widget as given in the *ID* field in the Glade app and *clicked* is an example of a signal established for that widget. The file will be invoked if the action associated with the signal is detected (for this example, the button is clicked). A given widget may have several signals defined for it and, consequently, several files to handle the several signals.

16.1 Mumps GTK GUI App Builder Example



```
adjustment1      GtkAdjustment
textbuffer1      GtkTextBuffer
treeStore        GtkTreeStore
window           GtkWindow
  fixed1         GtkFixed
    clearStore   GtkButton
    toggle1      GtkToggleButton
    entry1       GtkEntry
    scrolledrWindow1  GtkScr...
    label1       GtkLabel
    check1       GtkCheckButton
    scrolledWindow2   GtkScro...
      tree1      GtkTreeView
        select1  GtkTreeSelect...
        col0     GtkTreeViewCol...
          col0r  GtkCellRender...
        col1     GtkTreeViewCol...
          col1r  GtkCellRender...
        col2     GtkTreeViewCol...
          col2r  GtkCellRender...
    findButton   GtkButton
    radio1       GtkRadioButton
    radio2       GtkRadioButton
    radio3       GtkRadioButton
    spin1        GtkSpinButton
    loadTree     GtkButton
    loadTable    GtkButton
    show         GtkToggleButton
```

```
ID: toggle1
Activatable / Actionable
  Related Action:
  Use Action Appearance:
  Action Name:
Button Attributes
  Border relief: Normal
  Child alignments:
    Horizontal: 0.50
    Vertical: 0.50
  [x] Focus on click   [ ] Active
  [ ] Inconsistent     [ ] Draw Indicator
Button Content
  [ ] Add custom content
  ( ) Stock Button
    Stock Button:
    Position: Left   [ ] Always show image
  (o) Label with optional image
    toggle example
    Image:            [ ] Use underline
    Position: Left    [ ] Always show image
```

Figure 92 Toggle Button Screen 1

In Figure 96 you see the a Glade layout page. The center panel is the layout for the on-screen app that is being built. Several widgets have been dragged and dropped into their positions from widgets available in dropdown menus that are accessible from the buttons at the top named *Toplevel*, *Containers*, *Control*, and *Display*.

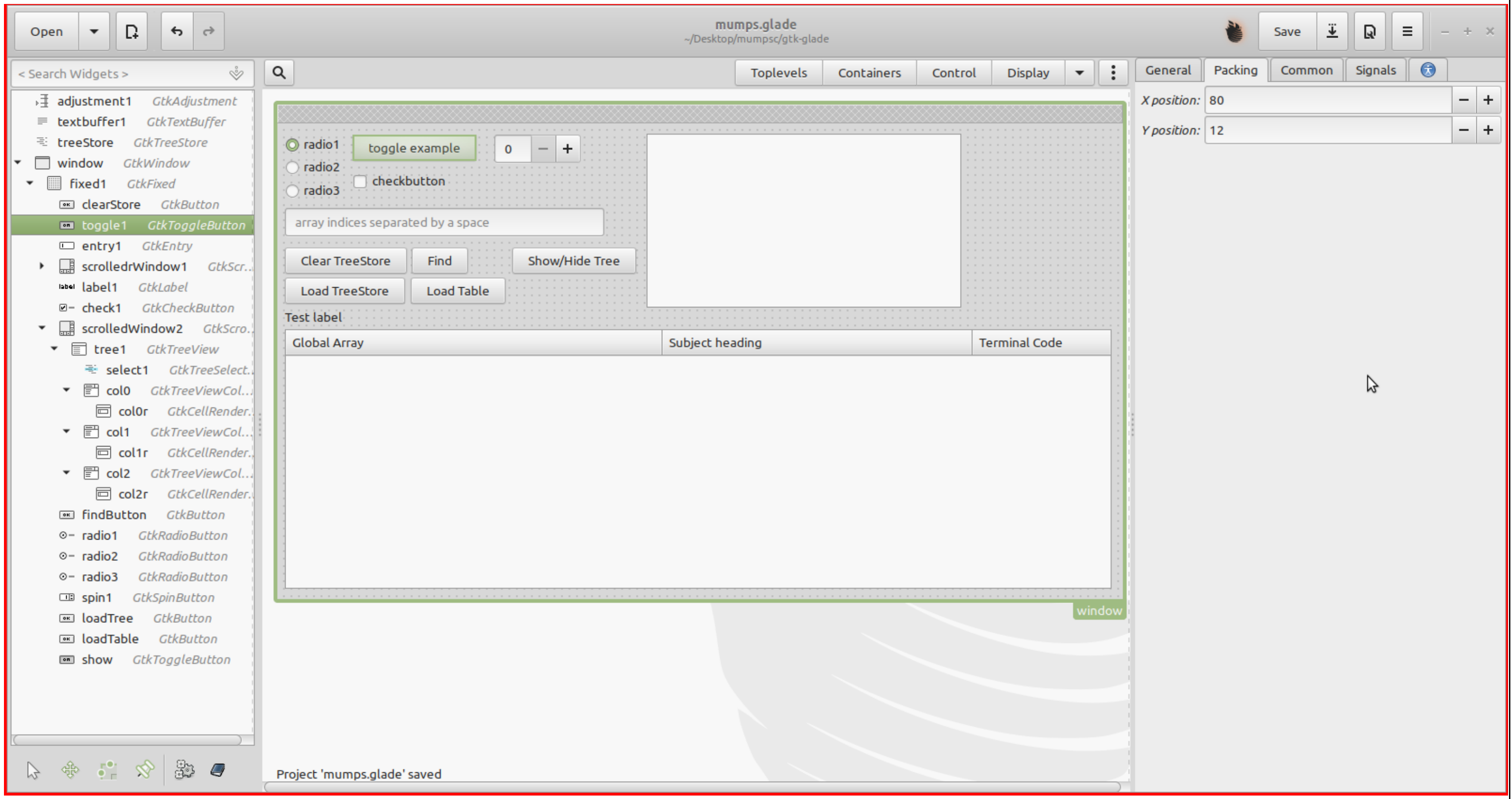
The leftmost panel contains the user assigned names (IDs) of the widgets along with their GTK data types (*GtkButton, GtkWindow, GtkEntry, etc.)*.

Some widgets are nested within others according to the display hierarchy. Thus, for example, the *GtkToggleButton* named *toggle1* is contained within the *GtkFixed* container named *fixed1* which in turn is contained within the *GtkWindow* named *window*.

The rightmost panel contains tabs. Each tab causes a panel to display of options and settings for the widget selected in the leftmost panel. In this case, the selected widget is the *toggle1* button which is highlighted in green in leftmost panel as well as it's visualization in the center panel.

As can be seen in panels 1 and 3, the ID of the widget is *toggle1* (user assigned), The widget data type is a *GtkToggleButton* (as seen in panel 1). The displayed contents of the toggle button (panel three, lower box) are *toggle example.*

Except for assigning the ID name of the widget and entering the text to appear in the button, the remainder of the options are defaults which are suitable for most ordinary applications.



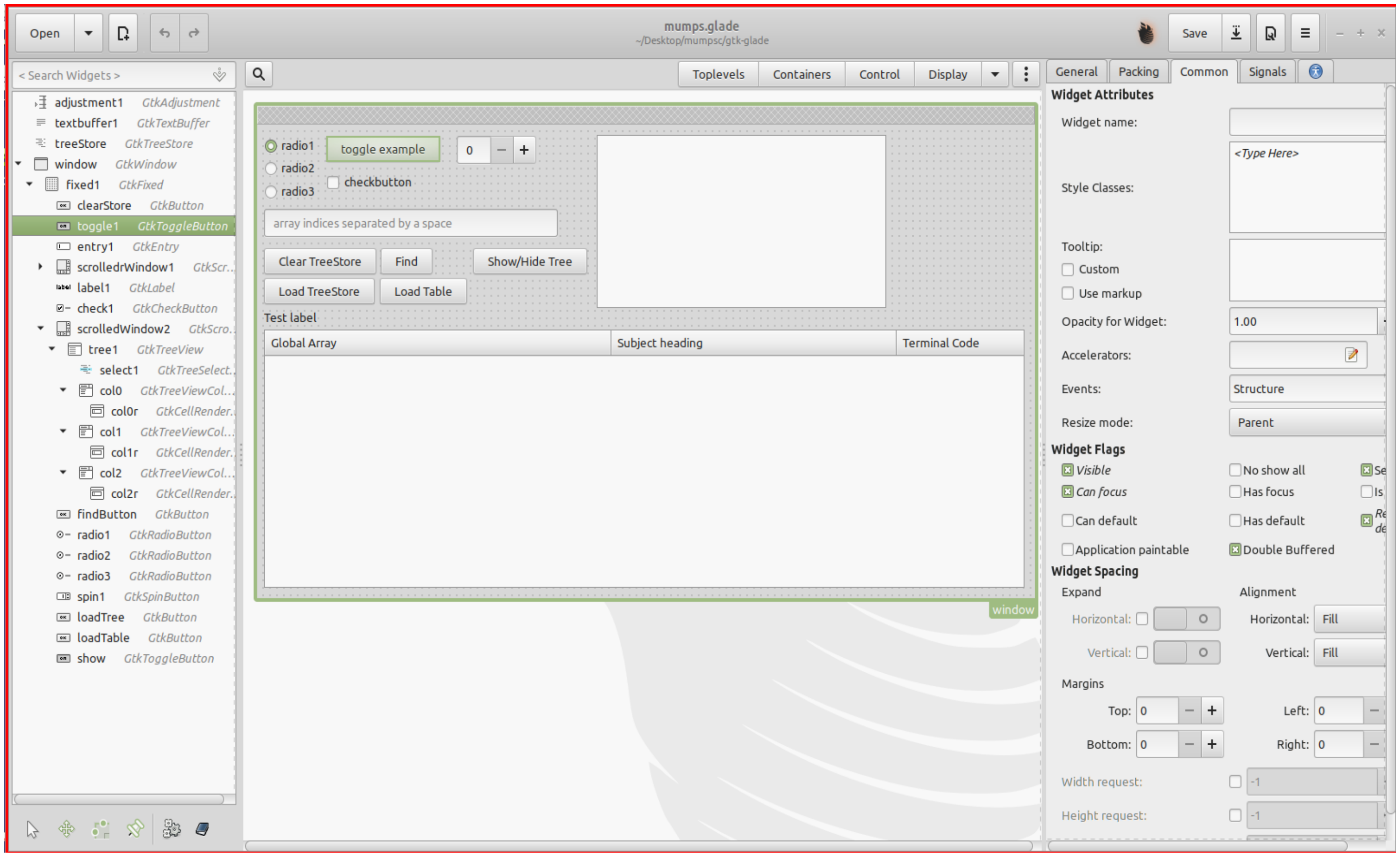
adjustment1 GtkAdjustment
textbuffer1 GtkTextBuffer
treeStore GtkTreeStore
window GtkWindow
fixed1 GtkFixed
clearStore GtkButton
toggle1 GtkToggleButton
entry1 GtkEntry
scrolledrWindow1 GtkScr...
label1 GtkLabel
check1 GtkCheckButton
scrolledWindow2 GtkScro...
tree1 GtkTreeView
select1 GtkTreeSelect...
col0 GtkTreeViewCol...
col0r GtkCellRender...
col1 GtkTreeViewCol...
col1r GtkCellRender...
col2 GtkTreeViewCol...
col2r GtkCellRender...
findButton GtkButton
radio1 GtkRadioButton
radio2 GtkRadioButton
radio3 GtkRadioButton
spin1 GtkSpinButton
loadTree GtkButton
loadTable GtkButton
show GtkToggleButton

X position: 80
Y position: 12

Project 'mumps.glade' saved

Figure 93 Toggle Button Screen 2

In Figure 93 the second tab (*Packing*) of panel 3 has been selected. This panel determines the location of the widget within the window. Changing these numbers moves the widget accordingly. The location (0,0) is the upper left corner of the container in which the widget appears.



Widget Attributes
Widget name:
Style Classes: <Type Here>
Tooltip:
☐ Custom
☐ Use markup
Opacity for Widget: 1.00
Accelerators:
Events: Structure
Resize mode: Parent
Widget Flags
☒ Visible ☐ No show all
☒ Can focus ☐ Has focus
☐ Can default ☐ Has default
☐ Application paintable ☒ Double Buffered
Widget Spacing
Expand
Horizontal: ☐
Vertical: ☐
Alignment
Horizontal: Fill
Vertical: Fill
Margins
Top: 0
Left: 0
Bottom: 0
Right: 0
Width request: ☐ -1
Height request: ☐ -1

Figure 94 Toggle Button Screen 3

In the third tab (*Common*) of panel 3 are many adjustments all of which are defaults except for the height and width settings. These determine the overall size of the button. In this case, the height and width request boxes have been unchecked which causes the button to be auto sized to fit the contained text.

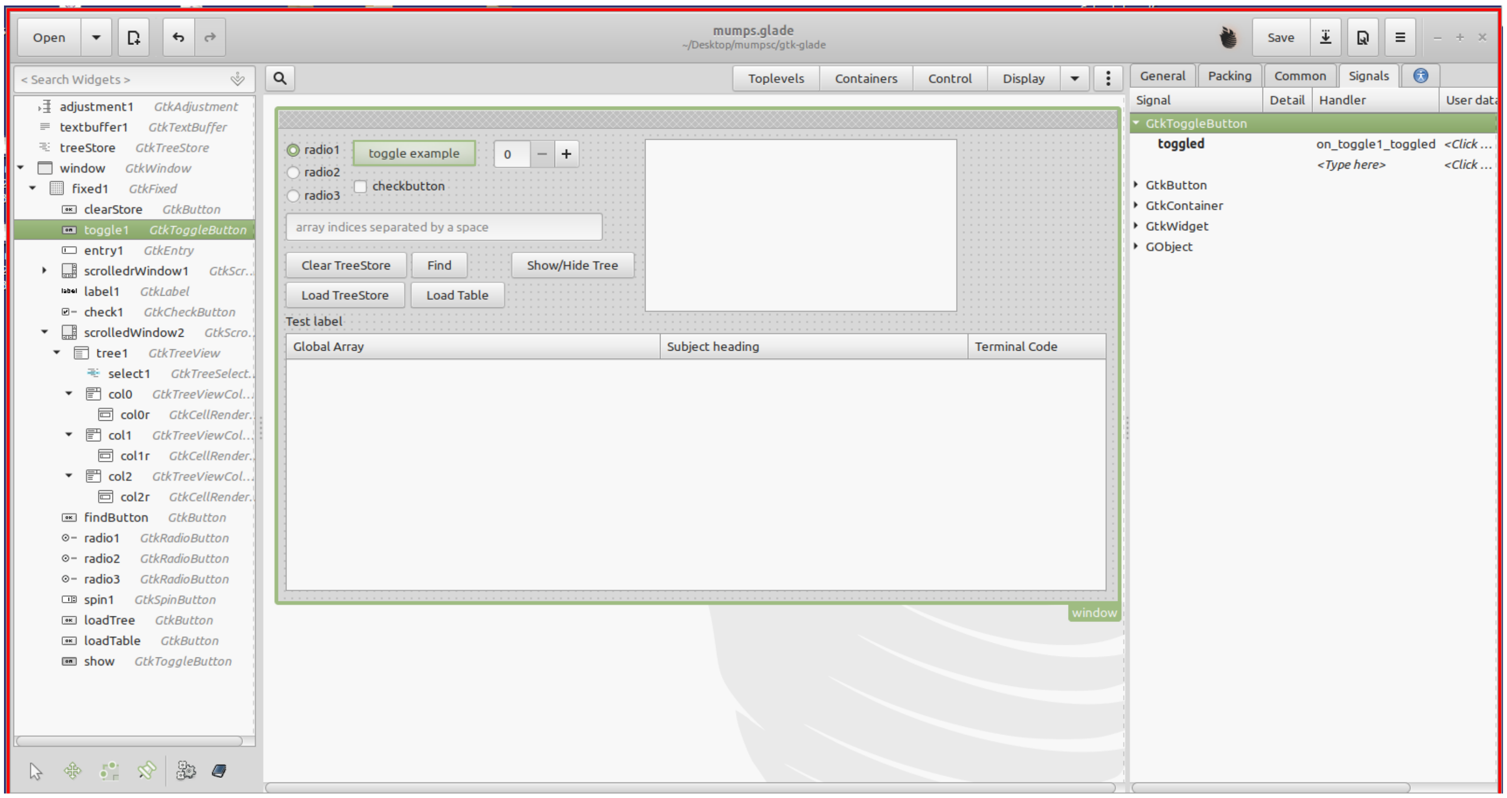


Figure 95 Toggle Button Screen 4

In Figure 95 we see the last tab (*Signals*) of panel 3. This is the panel where you select the signals to be emitted for actions on the widget. Since this is a toggle button, the usual primary user action is to click the button using the left mouse button[53]. This action emits the *toggled* signal.

The *Handler* column names the function that will process a signal (*on_toggle1_toggled* in this case). Ultimately, if a user clicks on this button, a call will be made to the Mumps program *on.toggle1.toggled.mps* which will perform whatever action is intended.

53 Note: the user may also click the button with the center or right button as well as other actions. A left button click, however, is by far the most common.

If you want your program to process a particular signal, you enter the name of the routine to be called should the signal be emitted. In this case, the function named *on_toggle1_toggled* will be called if the button is clicked. Since this is a toggle button, the GTK GUI manager will cause the button to appear depressed or not depressed after successive clicks. Your function can determine the state of the button by calling a system function.

When you save a Glade layout, it is saved as an XML file with the extension *.glade*.

16.1.1 Building A Mumps App from The Glade XML File

The disk representation of a Glade design is a XML file. For purposes of building a Mumps program from this file, the file needs to be named *mumps.glade*. This is the file name that *extractWidgets.mps* looks for.

In the above we highlighted the *toggl1* toggle button. The Glade XML fragment for that button looks like that shown in Figure 96.

```
        <child>
          <object class="GtkToggleButton" id="toggle1">
            <property name="label" translatable="yes">toggle example</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="receives_default">True</property>
            <signal name="toggled" handler="on_toggle1_toggled" swapped="no"/>
          </object>
          <packing>
            <property name="x">80</property>
            <property name="y">12</property>
          </packing>
        </child>
```

Figure 96 Sample Glade XML

The code in Figure 96 is part of the larger Glade file which is 299 lines in length. The XML tells us that the name of the widget (*toggle1*), its data type (*GtkToggleButton*), its text label contents (*toggle example*), any signals it emits (*toggled*) and the name of the signal handlers (*on_toggle1_toggled*). It also gives the location of the button on the app window and other information concerning its appearance and performance.

```
#!/usr/bin/mumps

# Jan 11, 2021

 set C=1

 write !!,"Mumps GTK Application Builder ",$zd,!!

 open 1:"gtk1.h,new"
 open 2:"gtk2.h,new"
 open 3:"gtk3.h,new"

 use 1 write "#include <gtk/gtk.h>",!
 use 1 write "#include <gtk/gtkx.h>",!
 use 1 write "GtkBuilder *builder;",!

 use 2 write "FILE *f=fopen(""gtk4.h"",""w"");",!
 use 2 write "gtk_init(&argc, &argv);",!
 use 2 write "builder = gtk_builder_new_from_file (""mumps.glade"");",!

 for  do
 . use 5
 . read line
 . if '$test break
 . set i=$find(line,"object class=")
 . if i'=0 do
 .. set p1=$extract(line,i,255)
 .. set widget=$piece(p1,"""",2)
 .. set id=$p(p1,"""",4)

 .. if widget="GtkCellRendererText" use 2 write "GtkCellRenderer *",id,";",!
 .. else use 1 write widget," *",id,";",!
 .. use 2 write !,id,"="

 .. if widget="GtkWindow" use 2 write "GTK_WINDOW("
 .. if widget="GtkFixed" use 2 write "GTK_FIXED("
 .. if widget="GtkButton" use 2 write "GTK_BUTTON("
 .. if widget="GtkAdjustment" use 2 write "GTK_ADJUSTMENT("
 .. if widget="GtkRadioButton" use 2 write "GTK_RADIO_BUTTON("
 .. if widget="GtkSpinButton" use 2 write "GTK_SPIN_BUTTON("
 .. if widget="GtkLabel" use 2 write "GTK_LABEL("
 .. if widget="GtkToggleButton" use 2 write "GTK_TOGGLE_BUTTON("
 .. if widget="GtkEntry" use 2 write "GTK_ENTRY("
 .. if widget="GtkViewport" use 2 write "GTK_VIEWPORT("
 .. if widget="GtkScrolledWindow" use 2 write "GTK_SCROLLED_WINDOW("
 .. if widget="GtkTextView" use 2 write "GTK_TEXT_VIEW("
 .. if widget="GtkTextBuffer" use 2 write "GTK_TEXT_BUFFER("
 .. if widget="GtkCheckButton" use 2 write "GTK_CHECK_BUTTON("
 .. if widget="GtkTreeView" use 2 write "GTK_TREE_VIEW("
 .. if widget="GtkTreeSelection" use 2 write "GTK_TREE_SELECTION("
 .. if widget="GtkTreeViewColumn" use 2 write "GTK_TREE_VIEW_COLUMN("
 .. if widget="GtkCellRendererText" use 2 write "GTK_CELL_RENDERER("
 .. if widget="GtkListStore" use 2 write "GTK_LIST_STORE("
 .. if widget="GtkTreeStore" use 2 write "GTK_TREE_STORE("

 .. use 2 write "gtk_builder_get_object(builder,""",id,"""));",!
 .. if C do
 ... use 2 write "{ char tmp[128]; sprintf(tmp,""%p"", ",id,");",!
 ... use 2 write " SymPut(""",id,""",tmp); "
 ... write "fprintf(f,"" set ",id,"=\""%s\""\n"",tmp); }",!

 .. if widget="GtkCellRendererText" do
 ... set c=$p(id,"col",2) set c=$p(c,"r",1)
 ... use 2 write !,"gtk_tree_view_column_add_attribute("
 ... write $p(id,"r",1),",",id,",""text"",",c,");",!

 . set i=$find(line,"signal name=")
 . if i'=0 do
 .. set sig=$piece(line,"""",4,4)
 .. if C set sig1=$translate(sig,"_",".")_".mps"
 .. if 'C set sig1=$translate(sig,"_",".")_".c"
 .. if C use 3 write "extern ""C"" void ",sig,"(GtkWidget *w) ",!
 .. if 'C use 3 write "#include """,sig1,"""",!
 .. if C do
 ... write "{struct MSV * Ptr = AllocSV(); char tmp[512]; ",!
 ... write "sprintf(tmp,""set widget=\""%p\"" g ^",sig1,""",w); ",!
 ... write "Interpret((const char *) tmp, Ptr); free(Ptr);}",!

 .. if '$zfile(sig1) do
 ... use 5 write "Creating signal handler: ",sig1,!
 ... if 'C use 5 write "Creating signal handler: ",sig1,!
 ... set sig2=sig1_",new"
 ... open 4:sig2
 ... if C use 4 write "#!/usr/bin/mumps",!
 ... if 'C use 4 write "void ",sig,"(GtkWidget *w) {",!
 ... if C do
 .... use 4 write !,"#    Mumps GTK Signal Handler",!
 .... use 4 write !," do ^gtk4.h",!
```

```
.... use 4 write " write """,sig1,""",""" "",widget,!",!
... if 'C use 4 write " printf(""%s\n"", """,sig1,""");",!

... if C do
.... if widget="GtkTreeSelection" do
..... use 4 write " write $z~mdh~tree~selection~get~selected("_id_",0),!",!
... if 'C do
.... if widget="GtkTreeSelection" do
..... use 4 write "GtkTreeSelection *s;",!
..... write "GtkTreeModel *m;",!,"gchar *value;",!,"GtkTreeIter iter;",!
..... write "if (gtk_tree_selection_get_selected "
..... write (GTK_TREE_SELECTION(w), &m, &iter) == FALSE) return; ",!
..... write "gtk_tree_model_get(m, &iter, 0, &value,  -1); "
..... write "printf(""col 1 = %s\n"", value);",!

... if C do
.... if widget="GtkToggleButton" do
..... use 4 write " write $z~mdh~toggle~button~get~active("_id_"),!",!
... if 'C do
.... if widget="GtkToggleButton" do
..... use 4 write " printf(""toggled = %d\n"", "
..... write "gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(w)));",!

... if C do
.... if widget="GtkSpinButton" do
..... use 4 write " write $z~mdh~spin~button~get~value("_id_"),!",!
... if 'C do
.... if widget="GtkSpinButton" do
..... use 4 write " printf(""spin = %f\n"", "
..... write "gtk_spin_button_get_value(GTK_SPIN_BUTTON(w)));",!

... if C do
.... if widget="GtkCheckButton" do
..... use 4 write " write $z~mdh~toggle~button~get~active("_id_"),!",!
... if 'C do
.... if widget="GtkCheckButton" do
..... use 4 write " printf(""active = %d\n"", "
..... write "gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(w)));",!

... if C do
.... if widget="GtkEntry" do
..... use 4 write " write $z~mdh~entry~get~text("_id_"),!",!
... if 'C do
.... if widget="GtkEntry" do
```

```
 ..... use 4 write " printf(""%s\n"", gtk_entry_get_text(GTK_ENTRY(w)));",!

 ... if 'C use 4 write " }",!

 ... close 4

 use 2 write !,"g_signal_connect(window, ""destroy"", "
 write "G_CALLBACK(gtk_main_quit), NULL);",!
 use 2 write "gtk_builder_connect_signals(builder, NULL);",!
 use 2 write "gtk_widget_show(GTK_WIDGET(window));",!

 use 2 write "fclose(f);",!
# use 2 write "gtk_main();",!

 close 1
 close 2
 close 3
```

Program 53 Processing Glade XML

The distro program *extractWidgets.mps* (see Program 53) reads the XML file. It generates files that are used to compile and service an application. These files will be included into a template file named *gtk.mps* in order to build the GUI functional code. The files to be built are:

16.1.1.1 gtk1.h

This file contains C declarations for all the widgets defined in the XML file. It also includes the relevant GTK header files. In the case of the *toggle1* widget, the line:

```
GtkToggleButton *toggle1;
```

appears, among several others. The contents of *gtk1.h* will be included into the template fil *gtk.mps*

16.1.1.2 gtk2.h

This file contains code that will invoke a Mumps signal handler (see below) for each signal emitted for a widget. In the case of the *toggle1* widget, this code looks the code in Figure 97.

```
    toggle1=GTK_TOGGLE_BUTTON(gtk_builder_get_object(builder,"toggle1"));
    { char tmp[128]; sprintf(tmp,"%p", toggle1);
     SymPut("toggle1",tmp); fprintf(f," set toggle1=\"%s\"\n",tmp); }
```

Figure 97 Example *gtk2.h* Code

The code fragment in Figure 97 will be compiled into the base template program *gtk.mps.* For each widget, there will be a section of code similar to that in Figure 97 that builds the internal data structure and screen representation associated with the widget by means of the GTK function *gtk_builder_object().*

The function *gtk_builder_object* function parses the *mumps.glade* XML file in oder to build data structures for the widgets of the application. For the widget *toggle1,* the function returns a *GtkToggleButton* pointer to the data structure for *toggle1* (note: the names of the widgets and the internal pointers as usually the same. Both are named *toggle1* in this case).

The code fragment also will create a string value of the pointer which will be stored in the Mumps symbol table (*SymPut()*) and a *set* command similar to: *set toggle1=0x123456* is written to the file *gtk4.mps*. Note that the file *gtk4.mps* is created when the template file *gtk.mps* is executed.

16.1.1.3 gtk3.h

This file contains the entry points for the signal handlers. These are written in written in C and are used to invoke the corresponding Mumps signal handler programs which will actually process the signals. The code for the *toggle1* widget looks like:

```
    extern "C" void on_toggle1_toggled(GtkWidget *w)
    {struct MSV * Ptr = AllocSV(); char tmp[512];
    sprintf(tmp,"set widget=\"%p\" g ^on.toggle1.toggled.mps",w);
    Interpret((const char *) tmp, Ptr); free(Ptr);}
```

Figure 98 Example *gtk3.h* Code

This fragment establishes the C signal handler entry point *on_toggle1_toggled()*, creates an instance of the Mumps state vector (*MSV *Ptr*), creates a string consisting of Mumps *set* and *goto* (g) *commands* with the string value of the widget pointer *w* as the right hand side of the *set* command.

The first line specifies that the calling conventions for this function will follow C language rules. This is because the Mumps compiler and interpreter are actually C++ programs while the basic GTK library is written in C. The Mumps Compiler uses C++ conventions.

The subject of the *goto* command is a file named *^on.toggle1.clicked.mps* which will contain the Mumps code to process the signal. The name of any Mumps program created to process a signal will look similar to the original signal handler function entry point with the underscore characters replaced by decimal points. The underscore character may not be used in a Mumps name as the underscore character is an operator (concatenation) in Mumps.

Next, the signal handler invokes the mumps interpreter (*Interpret()*) which executes the commands in *tmp*.

16.1.1.4 gtk4.h

This file is created when the application is actually executed. The template file *gtk.mps* writes, for each widget, a Mumps set command that establishes the address of the data structure for the widget. In the case of the *toggle1* example, this looks like:

```
set toggle1="0x55ab6337e230"
```

When the Mumps signal handler is invoked, the file containing this information will be run by the Mumps signal handler thus giving the Mumps signal handler the memory references of all widgets in the application.

16.1.1.5 gtk.mps Template

This is the main routine that is compiled by the Mumps compiler. It will start the GTK GUI system. It looks like the code shown in Figure 99

```
# Jan 30, 2022
+ #include "gtk1.h"
        zmain
+ #include "gtk2.h"
        do ^gtk4.h
+ gtk_main();
        write "Goodbye!",!
        zexit
+ #include "gtk3.h"
```

Figure 99 *gtk.mps* Template

The lines that begin with a plus sign (+)are passed directly to the C++ compiler.

The function *gtk_main()* invokes the GTK main loop. Return is only made upon program termination.

The first *#include* brings in the global widget declarations (in C++). The second *#include* incorporates all the builder calls which creat the widgets on the screen along with their associated data structures. The third *#include* brings in the C signal handlers for all signals used by the widgets.

16.1.1.6 on.toggle1.toggled.mps

The actual Mumps signal handler created by *extractWidgets.mps*, named *on.toggle1.toggled.mps* looks like that shown in Figure 100.

```
#!/usr/bin/mumps

#       Mumps GTK Signal Handler

 do ^gtk4.h
 write "on.toggle1.toggled.mps"," ",widget,!
 write $z~mdh~toggle~button~get~active(toggle1),!
```

Figure 100 Example Mumps Signal Handller

The function *$z~mdh~toggle~button~get~active(toggle1)* returns 0 or 1 depending if the button is not depressed or depressed. In this case of the function, it’s Mumps reference (toggle1) was used but the variable *widget* is also present which contains a pointer to the data structure of the widget (*toggle1* in this case) which emitted the signal.

16.2 MDH Functions in Mumps

The following is a list of currently available GTK functions in Mumps.

16.2.1 $z~mdh~toggle~button~get~active(ToggleButtonReference)

Returns 0 if the button is inactive, 1 if active

16.2.2 $z~mdh~toggle~button~set~active(ToggleButtonReference,intVal)

Sets the button to active if intVal is 1, inactive if the value is 0.

16.2.3 $z~mdh~dialog~new~with~buttons(ParentWindowRef,dialog)

Raises a Gtk Dialog window displaying the contents of *dialog* with buttons **Yes** and **No**. Returns 1 if **Yes** is clicked; 0 if **No** is clicked; and -1 if the box is dismissed.

16.2.4 $z~mdh~entry~get~text(EntryReference)

Returns the current string contents of the referenced Entry box.

16.2.5 $z~mdh~entry~set~text(EntryReference,value)

Sets the contents of the named entry box to *value*.

16.2.6 $z~mdh~text~buffer~set~text(TextBufferReference,string)

Sets the contents of the referenced text buffer to the value of string.

16.2.7 $z~mdh~label~set~text(LabelReference,string)

Sets the text contents of the label referenced to string. Triggers a value changed signal.

16.2.8 $z~mdh~tree~selection~get~selected(TreeModelReference,column)

Returns value in designated column of referenced TreeModel.

16.2.9 $z~mdh~tree~store~clear(TreeStoreReference)

Clears (deletes) the contents of the referenced TreeStore.

16.2.10 $z~mdh~tree~level~add(TreeStoreReference,treeDepth,index,data[,...])

Add index at tree level treeeDepth to column 1 of TreeStore. Add additional data items in successive columns.

16.2.11 $z~mdh~spin~button~get~value(SpinButtonReference)

Returns the current value of the referenced SpinButton.

16.2.12 $z~mdh~spin~button~set~value(SpinButtonReference,number)

Sets the current value of the referenced spin button to number.

16.2.13 $z~mdh~widget~hide(widgetReference)

Hides the widget from view.

16.2.14 $z~mdh~widget~show(widgetReference)

Displays (un-hides) the widget.

16.3 GUI Based Retrieval Example

Included in the software package are routines to build GUI based retrieval models based on Glade, GTK, and Mumps. The code assumes the vector space database described above.

The example software, found in the *gtk-glade* sub-directory of the distribution, is designed to display the contents of the hyper-clusters of documents produced by the indexing software. It can be easily modified to display other aspects as well.

The software assumes that there are soft links from the *gtk-glade* directory to files in the main ISR directory. These links are:

1. key.dat
2. data.dat
3. ohsu.medline

If they are missing, please create them. The example GUI software depends on them to access the data and assumes that the user has executed *index.script* on the full 5,000 document database[54] using the original run-time parameters provided.

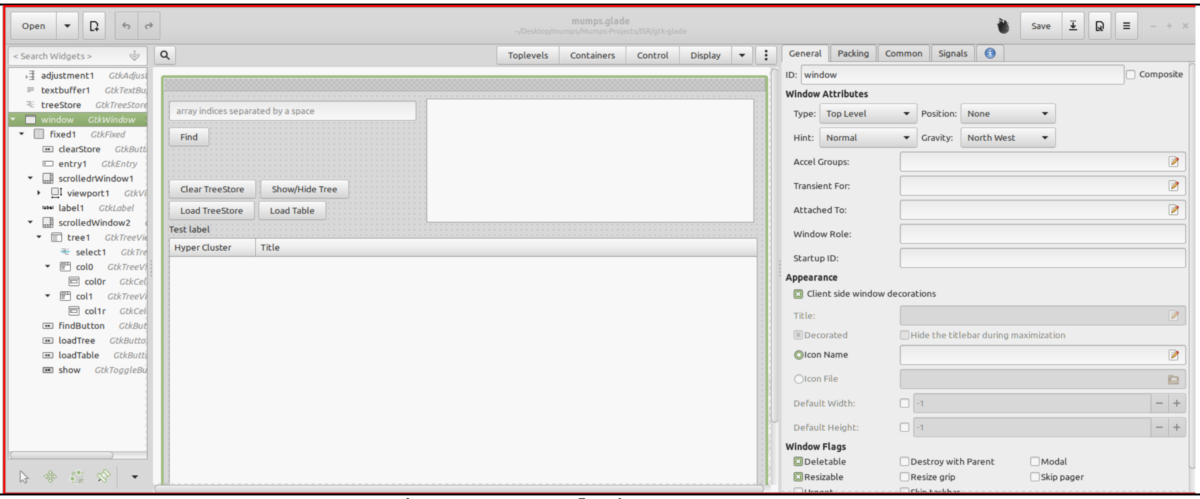
54 Note: smaller samples of the database may not produces sufficient clusters for the GUI to display.

The GUI example uses Glade which you should install if it is not already present[55]. Also, you will need the GTK development software. As on this writing, GTK3+ is the dominant version but GTK4 has been released.

GTK4 includes some additions and some name changes but the code presented here should be compatible with it. The examples presented here were developed with GTK3+.

You will need libgtk-3-0, libgtk-3-bin, libgtk-3-common, libgtk-3-dev, and libgtk3-doc or the most recent. These should already be installed when you installed the Mumps language processor.

In the distribution, there is a file named *mumps.glade* which contains the layout for the GUI. Double clicking on this file from a file browser should invoke Glade which will display the contents of the file. The file is actually an XML file containing information concerning widgets and their layout. Figure 101 shows what you should see when you invoke Glade on *mumps.glade*.



adjustment1 GtkAdjust
textbuffer1 GtkTextBu
treeStore GtkTreeStore
window GtkWindow
fixed1 GtkFixed
clearStore GtkButt
entry1 GtkEntry
scrolledrWindow1
viewport1 GtkVi
label1 GtkLabel
scrolledWindow2
tree1 GtkTreeVie
select1 GtkTre
col0 GtkTreeVi
col0r GtkCel
col1 GtkTreeVi
col1r GtkCel
findButton GtkBut
loadTree GtkButto
loadTable GtkButt
show GtkToggleBu

ID: window
Composite

Window Attributes

Type: Top Level Position: None
Hint: Normal Gravity: North West
Accel Groups:
Transient For:
Attached To:
Window Role:
Startup ID:

Appearance

☒ Client side window decorations
Title:
☒ Decorated ☐ Hide the titlebar during maximization
◉ Icon Name
○ Icon File
Default Width: ☐ -1
Default Height: ☐ -1

Window Flags

☒ Deletable ☐ Destroy with Parent ☐ Modal
☒ Resizable ☐ Resize grip ☐ Skip pager

Figure 101 Glade GUI Layout

Figure 101 contains:

55 The Mumps compiler and interpreter installation scripts will install Glade and GTK3.

1. In the left-most column, a hierarchical display of the widgets used in the layout beginning with *window* which will be the top-level GTK window. Components appearing in the window such as *tree1* (the large box on the bottom with the title bar containing the legends *Hyper Cluster* and *Title*).

2. In the central column, a visualization of the GTK widgets.

3. In the right-most column, attributes of the widget highlighted in green from the left-most column. These attributes include widget placement, size, and signals to be emitted upon user activity (such as a mouse click).

16.4 Generating An Application form the Glade File

After the Glade file has been constructed, it is passed as input to a Mumps program named *extractWidgets.mps* as was shown in Program 53.

Program 53 reads the Glade XML file and extracts from it descriptions of the functional widgets. From these descriptions, the program builds a collection of header (*.h*) and Mumps (*.mps*) files that constitute a skeleton of the application.

The header files are compiled with a Mumps template file using the Mumps compiler to a C++ program which serves as the main GTK signal handler while the *.mps* files become the actual signal handlers to be invoked by the main program when signals are received by the user.

The modified template file is shown in Program 54.

```
#!/usr/bin/mumps

#+ Copyright (C) 2022 by Kevin C. O'Kane
#+
#+ Kevin C. O'Kane, Ph.D.
#+ Professor Emeritus
#+ Computer Science Department
#+ University of Northern Iowa
#+ Cedar Falls, IA 50614-0507
#+ kc.okane@gmail.com
#+ http://www.cs.uni.edu/~okane
#+
#+ This program is free software; you can redistribute it and/or modify
#+ it under the terms of the GNU General Public License as published by
#+ the Free Software Foundation; either version 2 of the License, or
#+ (at your option) any later version.
#+
#+ This program is distributed in the hope that it will be useful,
#+ but WITHOUT ANY WARRANTY; without even the implied warranty of
#+ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#+ GNU General Public License for more details.
#+
#+ You should have received a copy of the GNU General Public License
#+ along with this program; if not, write to the Free Software
#+ Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

#      Feb 8, 2022

+ #include "gtk1.h"

       zmain

+ #include "gtk2.h"

       do ^gtk4.h

 set x="This is a test message"_$c(10)_"If it had been an actual message, it"
```

```
38.  set x=x_"would have said something important"
39.  set %=$z~mdh~text~buffer~set~text(textbuffer1,x)
40.
41. #      start the gtk_main() processor
42.
43. + gtk_main();
44.
45. #      gtk_main() returns when the program is finished
46.
47.        write "Goodbye!",!
48.        zexit
49.
50. + #include "gtk3.h"
```

Program 54 Mumps GUI Template File

The Mumps Compiler compiles Mumps programs to C++ files and these C++ files are compiled to executable binary.

Several lines of the code in Program 54 are actually C++ code that is passed directly to C++ program being constructed. These are the lines which begin with a plus (+) sign.

The template file works as follows:

The *include* on line 29 inserts into the output C++ program a collection of global GTK widget declarations. These were generated from the widgets in the Glade file by Program 53 and have the same names as the corresponding widgets. Program 55 has an example for the Glade layout shown in Figure 101. Note: the first two lines of Program 55 include standard system GTK support headers. The data types of the widget pointers are defined in the GTK documentation.

```
#include <gtk/gtk.h>
#include <gtk/gtkx.h>
GtkBuilder *builder;
GtkAdjustment *adjustment1;
GtkTextBuffer *textbuffer1;
GtkTreeStore *treeStore;
GtkWindow *window;
GtkFixed *fixed1;
GtkButton *clearStore;
GtkEntry *entry1;
GtkScrolledWindow *scrolledrWindow1;
GtkViewport *viewport1;
GtkTextView *textview1;
GtkLabel *label1;
GtkScrolledWindow *scrolledWindow2;
GtkTreeView *tree1;
GtkTreeSelection *select1;
GtkTreeViewColumn *col0;
GtkTreeViewColumn *col1;
GtkButton *findButton;
GtkButton *loadTree;
GtkButton *loadTable;
GtkToggleButton *show;
```

Program 55 *gtk1.h* Widget Declarations

Line 31 of Program 54 tells the Mumps compiler to begin a *main()* function.

Line 33 includes the file *gtk2.h* which was constructed by Program 53. It consists of C++ code. The contents of *gtk2.h* are shown in Program 56.

```
FILE *f=fopen("gtk4.h","w");
gtk_init(&argc, &argv);
builder = gtk_builder_new_from_file ("mumps.glade");

adjustment1=GTK_ADJUSTMENT(gtk_builder_get_object(builder,"adjustment1"));
{ char tmp[128]; sprintf(tmp,"%p", adjustment1);
 SymPut("adjustment1",tmp); fprintf(f," set adjustment1=\"%s\"\n",tmp); }

textbuffer1=GTK_TEXT_BUFFER(gtk_builder_get_object(builder,"textbuffer1"));
{ char tmp[128]; sprintf(tmp,"%p", textbuffer1);
 SymPut("textbuffer1",tmp); fprintf(f," set textbuffer1=\"%s\"\n",tmp); }

treeStore=GTK_TREE_STORE(gtk_builder_get_object(builder,"treeStore"));
{ char tmp[128]; sprintf(tmp,"%p", treeStore);
 SymPut("treeStore",tmp); fprintf(f," set treeStore=\"%s\"\n",tmp); }

window=GTK_WINDOW(gtk_builder_get_object(builder,"window"));
{ char tmp[128]; sprintf(tmp,"%p", window);
 SymPut("window",tmp); fprintf(f," set window=\"%s\"\n",tmp); }

fixed1=GTK_FIXED(gtk_builder_get_object(builder,"fixed1"));
{ char tmp[128]; sprintf(tmp,"%p", fixed1);
 SymPut("fixed1",tmp); fprintf(f," set fixed1=\"%s\"\n",tmp); }

clearStore=GTK_BUTTON(gtk_builder_get_object(builder,"clearStore"));
{ char tmp[128]; sprintf(tmp,"%p", clearStore);
 SymPut("clearStore",tmp); fprintf(f," set clearStore=\"%s\"\n",tmp); }

entry1=GTK_ENTRY(gtk_builder_get_object(builder,"entry1"));
{ char tmp[128]; sprintf(tmp,"%p", entry1);
 SymPut("entry1",tmp); fprintf(f," set entry1=\"%s\"\n",tmp); }

scrolledrWindow1=GTK_SCROLLED_WINDOW(gtk_builder_get_object(builder,"scrolledrWindow1"));
{ char tmp[128]; sprintf(tmp,"%p", scrolledrWindow1);
 SymPut("scrolledrWindow1",tmp); fprintf(f," set scrolledrWindow1=\"%s\"\n",tmp); }

viewport1=GTK_VIEWPORT(gtk_builder_get_object(builder,"viewport1"));
{ char tmp[128]; sprintf(tmp,"%p", viewport1);
 SymPut("viewport1",tmp); fprintf(f," set viewport1=\"%s\"\n",tmp); }

textview1=GTK_TEXT_VIEW(gtk_builder_get_object(builder,"textview1"));
{ char tmp[128]; sprintf(tmp,"%p", textview1);
 SymPut("textview1",tmp); fprintf(f," set textview1=\"%s\"\n",tmp); }
```

```
label1=GTK_LABEL(gtk_builder_get_object(builder,"label1"));
{ char tmp[128]; sprintf(tmp,"%p", label1);
 SymPut("label1",tmp); fprintf(f," set label1=\"%s\"\n",tmp); }

scrolledWindow2=GTK_SCROLLED_WINDOW(gtk_builder_get_object(builder,"scrolledWindow2"));
{ char tmp[128]; sprintf(tmp,"%p", scrolledWindow2);
 SymPut("scrolledWindow2",tmp); fprintf(f," set scrolledWindow2=\"%s\"\n",tmp); }

tree1=GTK_TREE_VIEW(gtk_builder_get_object(builder,"tree1"));
{ char tmp[128]; sprintf(tmp,"%p", tree1);
 SymPut("tree1",tmp); fprintf(f," set tree1=\"%s\"\n",tmp); }

select1=GTK_TREE_SELECTION(gtk_builder_get_object(builder,"select1"));
{ char tmp[128]; sprintf(tmp,"%p", select1);
 SymPut("select1",tmp); fprintf(f," set select1=\"%s\"\n",tmp); }

col0=GTK_TREE_VIEW_COLUMN(gtk_builder_get_object(builder,"col0"));
{ char tmp[128]; sprintf(tmp,"%p", col0);
 SymPut("col0",tmp); fprintf(f," set col0=\"%s\"\n",tmp); }
GtkCellRenderer *col0r;

col0r=GTK_CELL_RENDERER(gtk_builder_get_object(builder,"col0r"));
{ char tmp[128]; sprintf(tmp,"%p", col0r);
 SymPut("col0r",tmp); fprintf(f," set col0r=\"%s\"\n",tmp); }

gtk_tree_view_column_add_attribute(col0,col0r,"text",0);

col1=GTK_TREE_VIEW_COLUMN(gtk_builder_get_object(builder,"col1"));
{ char tmp[128]; sprintf(tmp,"%p", col1);
 SymPut("col1",tmp); fprintf(f," set col1=\"%s\"\n",tmp); }
GtkCellRenderer *col1r;

col1r=GTK_CELL_RENDERER(gtk_builder_get_object(builder,"col1r"));
{ char tmp[128]; sprintf(tmp,"%p", col1r);
 SymPut("col1r",tmp); fprintf(f," set col1r=\"%s\"\n",tmp); }

gtk_tree_view_column_add_attribute(col1,col1r,"text",1);

findButton=GTK_BUTTON(gtk_builder_get_object(builder,"findButton"));
{ char tmp[128]; sprintf(tmp,"%p", findButton);
 SymPut("findButton",tmp); fprintf(f," set findButton=\"%s\"\n",tmp); }

loadTree=GTK_BUTTON(gtk_builder_get_object(builder,"loadTree"));
{ char tmp[128]; sprintf(tmp,"%p", loadTree);
```

```
 SymPut("loadTree",tmp); fprintf(f," set loadTree=\"%s\"\n",tmp); }

loadTable=GTK_BUTTON(gtk_builder_get_object(builder,"loadTable"));
{ char tmp[128]; sprintf(tmp,"%p", loadTable);
 SymPut("loadTable",tmp); fprintf(f," set loadTable=\"%s\"\n",tmp); }

show=GTK_TOGGLE_BUTTON(gtk_builder_get_object(builder,"show"));
{ char tmp[128]; sprintf(tmp,"%p", show);
 SymPut("show",tmp); fprintf(f," set show=\"%s\"\n",tmp); }

g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);
gtk_builder_connect_signals(builder, NULL);
gtk_widget_show(GTK_WIDGET(window));
fclose(f);
```

Program 56 *gtk2.h* Signal Handler Connections

For each widget detected by Program 53, the code in *gtk2.h* (Program 56) will:

1. Insert a line of C++ into the template program that, when the program is run, will connect each GTK widget declared in Program 55 to the details from the Glade XML file describing the widget and its location.
2. Insert for each widget a line of C++ code into the template program which, at run-time, will place the hexadecimal address of the widget into the Mumps run-time symbol table.
3. Create and append to a file named *gtk4.h* a line of Mumps code giving in hexadecimal the address of each widget.
4. Finally, Program 56 will place into the C++ template program lines of code that set up how the program will terminate in response to a *destroy* signal, connect the signal handlers to the GTK environment, and tell the GTK environment to display the main window.

Next, at run-time, on line 35, the template program will execute the file *gtk4.h* which was generated by *gtk2.h*. The *gtk4.h* for this example is shown in Program 57. Note: this file does not exist until the compiled template program is executed. Also note that the addresses shown will be different on your machine.

```
set adjustment1="0x562c94376ac0"
set textbuffer1="0x562c9436f490"
set treeStore="0x562c94378cf0"
set window="0x562c9438c260"
set fixed1="0x562c94340270"
set clearStore="0x562c944c9180"
set entry1="0x562c94556290"
set scrolledrWindow1="0x562c94598310"
set viewport1="0x562c942ffa60"
set textview1="0x562c94556540"
set label1="0x562c942ffc10"
set scrolledWindow2="0x562c94598650"
set tree1="0x562c945ce3e0"
set select1="0x562c945aef50"
set col0="0x562c945c69d0"
set col0r="0x562c945e3120"
set col1="0x562c945c6b70"
set col1r="0x562c945e3250"
set findButton="0x562c944c96c0"
set loadTree="0x562c944c9880"
set loadTable="0x562c944c9a40"
set show="0x562c945f0290"
```

Program 57 *gtk4.h*

Lines 37 through 38 of Program 54 initialize widget *textview1* (box at upper right of the layout).

Lines 43 through 48 of Program 54 start the GTK main loop and, when the program terminates (i.e., *gtk_main()* returns), exits.

Line 50 of Program 54 includes the file *gtk3.h* which contains the C++ entry points for the signal handers that were detected by Program 53. The file for this example is shown in Program

```
extern "C" void on_clearStore_clicked(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.clearStore.clicked.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
extern "C" void on_entry1_changed(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.entry1.changed.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
extern "C" void on_select1_changed(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.select1.changed.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
extern "C" void on_findButton_clicked(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.findButton.clicked.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
extern "C" void on_loadTree_clicked(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.loadTree.clicked.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
extern "C" void on_loadTable_clicked(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.loadTable.clicked.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
extern "C" void on_show_toggled(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.show.toggled.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
```

Program 58 *gtk3.h* Signal Handler Entry Points

For each signal detected in the Glade file, there must exist a signal handler.

Since the template program is in C++ but the actual signal handlers will be invoked as C programs (different subroutine linkage conventions), the C++ program is advised that the linked programs use C linkage. The lines that begin with: *extern "C"* accomplish this. Each instance of the *extern* attribute constitutes a signal handler function.

For example, the signal handler to handle mouse clicks on the layout button Clear TreeStore is:

```
extern "C" void on_clearStore_clicked(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.clearStore.clicked.mps",w);
      Interpret((const char *) tmp, Ptr); free(Ptr);}
```

The signal handler is invoked if the GUI user clicks the Clear TreeStore button. The name of the signal handler, *on_clear_Store*, was provided by the user when the Glade layout was constructed (these names are suggested by the Glade program although the user can choose any name).

When a signal handler is invoked, a function is called that allocates an interpreter stack frame, constructs a line of Mumps code:

set widget="xxx" g ^on.clearStore.clicked.mps

where *xxx* is inserted at run-time, invokes the Mumps interpreter passing to it the line of code constructed. Upon return, the stack frame is freed and the function exits.

The second part of the constructed line, *g ^on.clearStore.clicked.mps*, tells the Mumps interpreter to execute the program in *on.clearStore.clicked.mps*.

When the Glade XML file was scanned by *extractWidgets.mps* (Program 53), a program of this name was constructed with only a write statement giving its identity. Note: if a program by that name already exists, *extractWidgets.mps* will not over-write it.

It is in these Mumps programs that the user will place the code, in Mumps, that will process the signals. In the case of *on.clearStore.clicked.mps* Program 59 gives an example.

```
#!/usr/bin/mumps

#      Mumps GTK Signal Handler

 do ^gtk4.h
 write "on.clearStore.clicked.mps"," ",widget,!
 set i=$z~mdh~tree~store~clear(treeStore)
```

Program 59 Mumps Signal Handler *on.clearStore.clicked.mps*

In Program 59, the invocation of *gtk4.h* establishes the memory addresses of the widgets. The next line displays to the user terminal a message saying the program has been invoked. Note: the value of the Mumps variable *widget* was established by the C++ handler entry point in the *set* command.

The final line invokes a GTK function which clears the contents of the TreeStore.

A more extensive example is shown in Programs 60 and 61.

```
#!/usr/bin/mumps

#      Mumps GTK Signal Handler

 do ^gtk4.h
 write "on.loadTree.clicked.mps"," ",widget,!
 set i=$z~mdh~tree~store~clear(treeStore)
 set tree="^mesh(0)" ; both of these work
 set tree="^h" ; both of these work
 do ^LoadTreeView.mps
```

Program 60 Signal Handler *on.loadTree.clicked.mps*

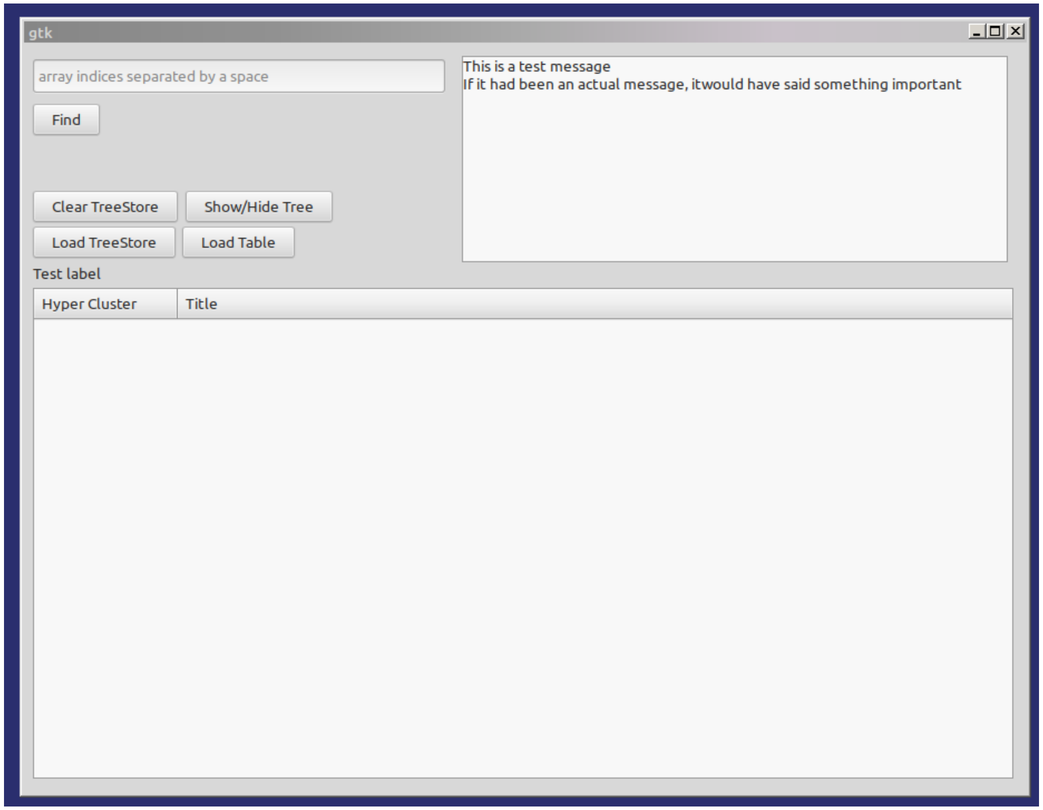
```
#!/usr/bin/mumps

#+ Copyright (C) 2022 by Kevin C. O'Kane
#+
#+ Kevin C. O'Kane, Ph.D.
#+ Professor Emeritus
#+ Computer Science Department
#+ University of Northern Iowa
#+ Cedar Falls, IA 50614-0507
#+ kc.okane@gmail.com
#+ http://www.cs.uni.edu/~okane
#+
#+ This program is free software; you can redistribute it and/or modify
#+ it under the terms of the GNU General Public License as published by
#+ the Free Software Foundation; either version 2 of the License, or
#+ (at your option) any later version.
#+
#+ This program is distributed in the hope that it will be useful,
#+ but WITHOUT ANY WARRANTY; without even the implied warranty of
#+ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#+ GNU General Public License for more details.
#+
#+ You should have received a copy of the GNU General Public License
#+ along with this program; if not, write to the Free Software
#+ Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

#       Feb 26, 2022

#       'tree' is the global array reference to load.
#       It may be a top level array reference or
#       a node within the global array.
#       treeStore is assumed to be empty.

#       how many indices in tree? (depth)

        set tmp=$query(@tree)   ; get the next node
        set depth=$qlength(tmp) ; how many indices in next node?

#       if tree reference greater that top level, add each
#       node of original query to tree.

       if depth>1 do
       . set tmp=tree          ; hold for future reference
       . for j=1:1:depth-1 do  ; next node depth
       .. set %=$z~mdh~tree~level~add(treeStore,j,tree,@tree)

#       set %=$z~mdh~tree~level~add(treeStore,treeDepth,col1,col2,col3)

        for  do
        . set tree=$query(@tree)
        . if tree="" break
        . set i=$qlength(tree)
       . w i," ",tree," ",@tree,!
#      . set %=$z~mdh~tree~level~add(treeStore,i,tree,@tree,$qsubscript(tree,i))
       . if i=1 set %=$z~mdh~tree~level~add(treeStore,i,"HyperCluster "_$qsubscript(tree,1),@tree)
       . if i=2 set %=$z~mdh~tree~level~add(treeStore,i,"Cluster "_$qsubscript(tree,i)," ")
       . if i=3 set %=$z~mdh~tree~level~add(treeStore,i,$qsubscript(tree,i),$e(^title($qsubscript(tree,i)),1,80))
       . if depth>1&(i=(depth-1)) if tmp'=tree break ; we've move beyond the query
```

Program 61 *LoadTreeView.mps*

16.5 Final Result

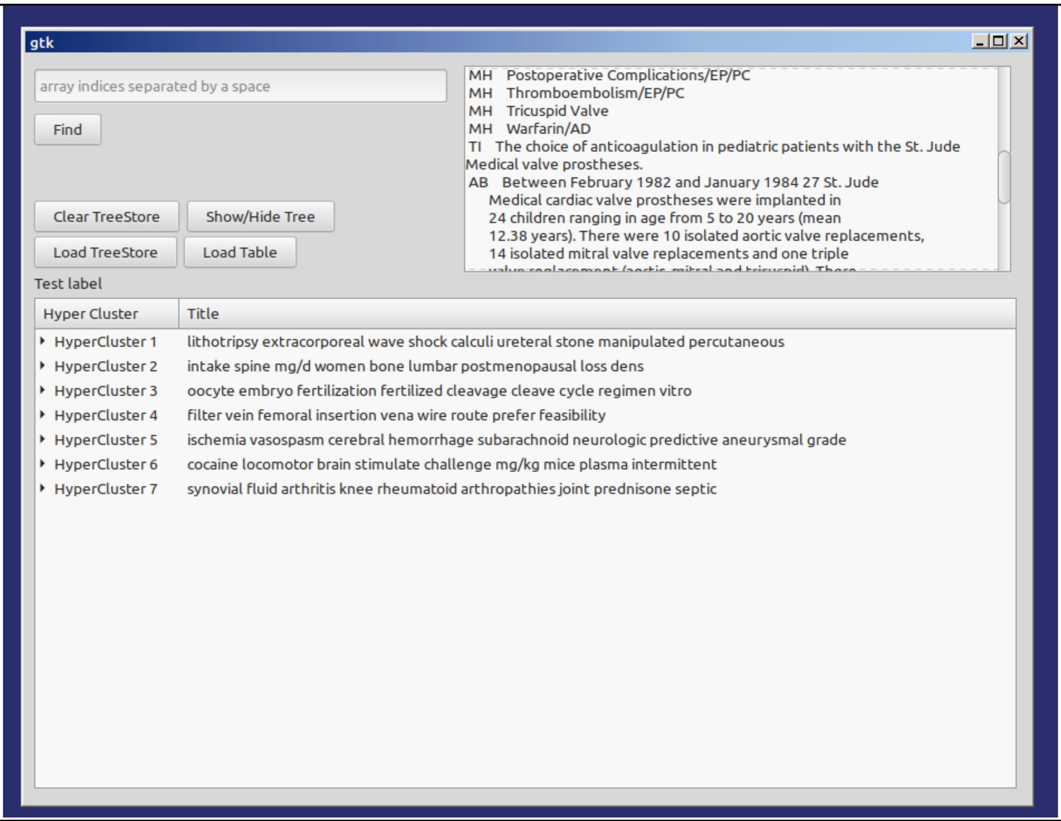
Figures 102, 103, and 104 show the final result when the program *gtk*is executed.

Upon clocking the Load TreeStore button the contents of the hyper-cluster database is loaded. Clicking on one of the sub-clusters reveals detailed documents and clicking on a document causes the origin *ohsu.medline* entry to be displayed in the upper right box.



This is a test message
If it had been an actual message, itwould have said something important

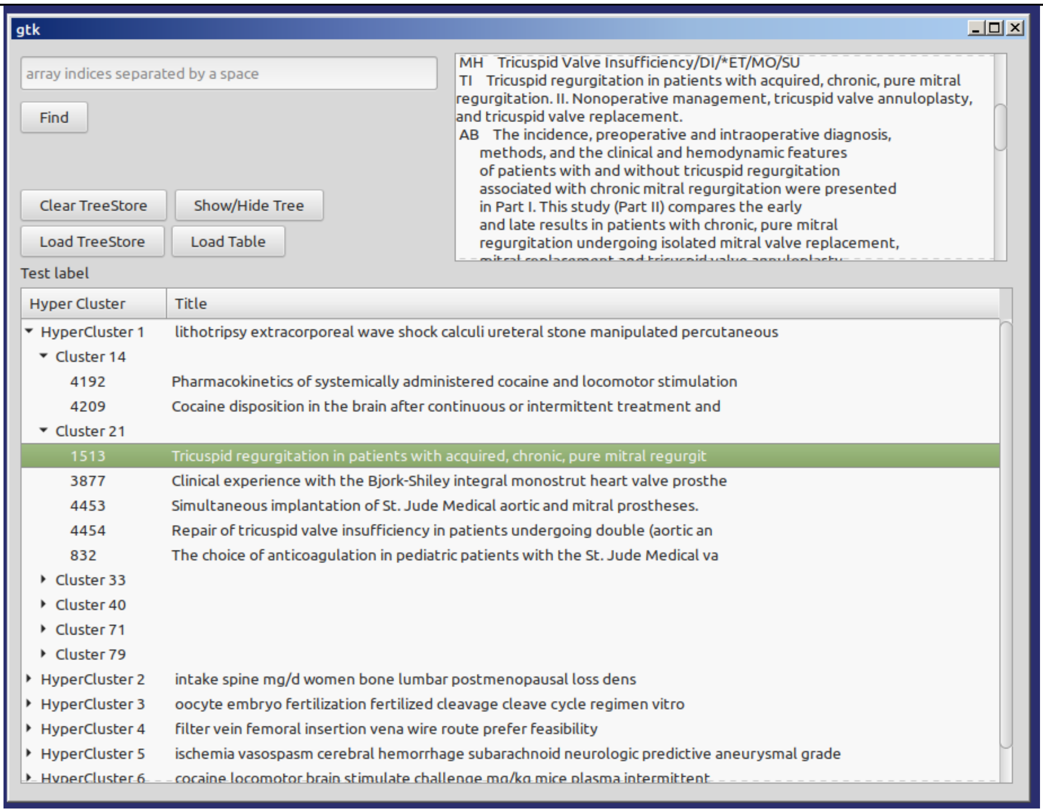
Figure 102 GUI Example 1



```
MH   Postoperative Complications/EP/PC
MH   Thromboembolism/EP/PC
MH   Tricuspid Valve
MH   Warfarin/AD
TI   The choice of anticoagulation in pediatric patients with the St. Jude
Medical valve prostheses.
AB   Between February 1982 and January 1984 27 St. Jude
     Medical cardiac valve prostheses were implanted in
     24 children ranging in age from 5 to 20 years (mean
     12.38 years). There were 10 isolated aortic valve replacements,
     14 isolated mitral valve replacements and one triple
```

Hyper Cluster	Title
HyperCluster 1	lithotripsy extracorporeal wave shock calculi ureteral stone manipulated percutaneous
HyperCluster 2	intake spine mg/d women bone lumbar postmenopausal loss dens
HyperCluster 3	oocyte embryo fertilization fertilized cleavage cleave cycle regimen vitro
HyperCluster 4	filter vein femoral insertion vena wire route prefer feasibility
HyperCluster 5	ischemia vasospasm cerebral hemorrhage subarachnoid neurologic predictive aneurysmal grade
HyperCluster 6	cocaine locomotor brain stimulate challenge mg/kg mice plasma intermittent
HyperCluster 7	synovial fluid arthritis knee rheumatoid arthropathies joint prednisone septic

Figure 103 GUI Example 2



gtk

array indices separated by a space

Find

Clear TreeStore | Show/Hide Tree

Load TreeStore | Load Table

```
MH   Tricuspid Valve Insufficiency/DI/*ET/MO/SU
TI   Tricuspid regurgitation in patients with acquired, chronic, pure mitral
regurgitation. II. Nonoperative management, tricuspid valve annuloplasty,
and tricuspid valve replacement.
AB   The incidence, preoperative and intraoperative diagnosis,
     methods, and the clinical and hemodynamic features
     of patients with and without tricuspid regurgitation
     associated with chronic mitral regurgitation were presented
     in Part I. This study (Part II) compares the early
     and late results in patients with chronic, pure mitral
     regurgitation undergoing isolated mitral valve replacement,
     mitral replacement and tricuspid valve annuloplasty
```

Test label

Hyper Cluster	Title
HyperCluster 1	lithotripsy extracorporeal wave shock calculi ureteral stone manipulated percutaneous
Cluster 14	
4192	Pharmacokinetics of systemically administered cocaine and locomotor stimulation
4209	Cocaine disposition in the brain after continuous or intermittent treatment and
Cluster 21	
1513	Tricuspid regurgitation in patients with acquired, chronic, pure mitral regurgit
3877	Clinical experience with the Bjork-Shiley integral monostrut heart valve prosthe
4453	Simultaneous implantation of St. Jude Medical aortic and mitral prostheses.
4454	Repair of tricuspid valve insufficiency in patients undergoing double (aortic an
832	The choice of anticoagulation in pediatric patients with the St. Jude Medical va
Cluster 33	
Cluster 40	
Cluster 71	
Cluster 79	
HyperCluster 2	intake spine mg/d women bone lumbar postmenopausal loss dens
HyperCluster 3	oocyte embryo fertilization fertilized cleavage cleave cycle regimen vitro
HyperCluster 4	filter vein femoral insertion vena wire route prefer feasibility
HyperCluster 5	ischemia vasospasm cerebral hemorrhage subarachnoid neurologic predictive aneurysmal grade
HyperCluster 6	cocaine locomotor brain stimulate challenge mg/kg mice plasma intermittent

Figure 104 GUI Example 3

17 Other Scrip Files

17.1.1 iterate.script

The file *iterate.script* is a *bash* script to run multiple instances of *index.script,* with multiple parameters, in parallel. It does this by building sub-directories and concurrently executing instances of *index.script* in these.

17.1.2 boolean-grep-0*.script

These are some *bash* script examples of how to perform Boolean searches on the database using common Linux utility software (*grep, egrep, sort, uniq, wc, etc*.).

17.1.3 Mesh.script

A *bash* script to index the document collection using the controlled vocabulary Medical Subjects Headings[56].

56 https://www.nlm.nih.gov/mesh

18 References

Aho 1990	Aho, Alfred V. (1990). "Algorithms for finding patterns in strings". In van Leeuwen, Jan. *Handbook of Theoretical Computer Science*, *Volume A: Algorithms and Complexity*. The MIT Press. pp. 255–300
ANSI 1995	American National Standards Institute, Inc. (1995). ANSI/MDC X11.4.1995 Information Systems. Programming Languages - M, American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
Blair 1996	Blair, D.C., STAIRS Redux: Thoughts on the STAIRS Evaluation, Ten Years after, *Journal American Society for Information Science,* Vol 47, No. 1, pp 2-22 (1996).
Barnett 1970	Barnett, G.O.,and Greenes, R.A., High level programming languages, *Computers and Biomedical Research*, Vol 3, pp 488 - 497 (1970)
Bowie 1976	Bowie, J., and Barnett, G. O., MUMPS: an economical and efficient time-sharing language for information management, *Computer Programs in Biomedicine*, Vol 6, pp 11 - 21 (1976).
Crouch 1988	Crouch, C., An analysis of approximate versus exact discrimination values, *Information Processing and Management*, Vol. 24, No. 1, pp. 5-16 (1988).
Frakes 1992	Frakes, W.B; and Baeza-Yates, R.; *Information Retrieval, Data Structures and Algorithms*, Prentice-Hall (Englewood Cliffs, NJ 1992).
Heaps 1978	Heaps, H.S., *Information Retrieval, Computational and Theoretical Aspects*, Academic Press (New York 1978).
Hersh 1994	Hersh WR, Buckley C, Leone TJ, Hickam DH, OHSUMED: An interactive retrieval evaluation and new large test collection for research, *Proceedings of the 17th Annual ACM SIGIR Conference*, 1994, 192-201.
Korfage 1997	Korfage, R., *Information Storage and Retrieval*, Wiley (New York, 1997).
Kowalski 2000	Kowalski, G.; and Maybury, M., *Information Storage and Retrieval Systems, Theory and Implementation, Second Edition*, Kluwer (Boston, 2000)

Luhn 1957	Luhn, H.P., A statistical approach to mechanized encoding and searching of literary information, *IBM Journal of Research and Development*, Vol 1, No 4, (1957)
Manning 1999	Manning, C.; Schütze, H., *Foundations of Statistical Natural Language Processing*, MIT Press: (1999).
Manning 2008	Manning, C.; Raghavan, P.; and Schütze, H., *Introduction to Information Retrieval*, Cambridge University Press (Cambridge 2008).
NCBI 2016	National Center for Biotechnology Information (NCBI), http://www.ncbi.nlm.nih.gov/
NIST 2000	National Institute of Standards and Technology, Text Retrieval Conference 9, http://trec.nist.gov/pubs/trec9/t9_proceedings.html (2000).
O'Kane 2008	O'Kane, K.C., *The Mumps Programming Language*, Amazon Createspace (2008).
Salton 1968	Salton, G., *Automatic Information Organization and Retrieval*, McGraw Hill (New York, 1968).
Salton 1971	Salton, G, ed.; *The SMART Retrieval System, Experiments in Automatic Document Processing*, Prentice-Hall (Englewood Cliffs, NJ, 1971).
Salton 1983	Salton, G.; and McGill, M.J., *Introduction to Modern Information Retrieval*, McGraw Hill; (New York, 1983).
Salton 1988	Salton, G., *Automatic Text Processing*, Addison-Wesley (Reading, 1988)
Salton 1992	Salton, G., The state of retrieval system evaluation, *Information Processing & Management*, Vol 28 No 4, pp. 441-449 (1992).
USNA 2007	U.S. National Archives, *The Soundex Indexing System*, http://www.archives.gov/research/census/soundex.htm (2007)
vanRijsbergen 1979	van Rijsbergen, C. J., *Information Retrieval*, Butterworths (London, 1979). http://www.dcs.gla.ac.uk/Keith/Preface.html
Willett 1985	Willett, P., An algorithm for calculation of exact term discrimination vales, *Information Processing and Management*, Vol 21, No. 3, pp 225-232 (1985).

Alphabetical Index