Indexing large collections of small text records for ranked retrieval

Robert K. France
Edward A. Fox*

Computing Center and Department of Computer Science*
Virginia Tech (Virginia Polytechnic Institute and State University)
Blacksburg VA 24061-0106

Abstract

The MARIAN online public access catalog system at Virginia Tech has been developed to apply advanced information retrieval methods and object-oriented technology to the needs of library patrons. We give a description of our data model, design, processing, data representations, and retrieval operation. By identifying objects of interest during the indexing process, storing them according to our "information graph" model, and applying weighting schemes that seem appropriate for this large collection of small text records, we hope to better serve user needs. Since every text word is important in this domain, we employ opportunistic matching algorithms and a mix of data structures to support searching, that will give good performance for a large campus community, even though MARIAN runs on a distributed collection of small workstations. An initial small experiment indicates that our new ad hoc weighting scheme is more effective than a more standard approach.

Introduction

The library is the archetype for information retrieval systems. On-line public access catalogs (OPACs) were an early application for automated IR systems, and have been widely studied (Hildreth, 1985, 1987; Yee, 1991). There has, however, been relatively little interpenetration between the OPAC community and that portion of the IR community concerned with text retrieval (Borgman, 1986), in part perhaps because library catalog records have so little text and so many other aspects. Nonetheless, there are many problems related to current OPACs that could be solved by advanced retrieval

techniques; we consider but five obvious ones. First, current OPACs tend to respond poorly to vague queries, retrieving either too few or too many works. Second, they provide few or no paths for users to move from a relevant work to other similar works --- they fail to provide a clear context in the space of works, and hence cannot support the type of simple browsing encouraged by library stacks. Third, by adopting the standard Boolean query paradigm, they make it difficult for most users to make use of fragmented recollections, such as when one vaguely remembers part of an author's name and parts of several subject descriptors. Fourth, they lack any understanding of English morphology, and so cannot help users who do not give the exact form of words in titles. Fifth, most current OPACs operate on expensive mainframes or minicomputers, even though a more cost effective solution would be to use (a small cluster of) workstations. The MARIAN library catalog system at Virginia Tech is designed to address these problems in a production system through the use of advanced information representation and retrieval techniques (Fox et al., 1993). This paper discusses the data models and indexing techniques that support the system.

The MARC record (Library of Congress, 1988a) is a specialized data structure for the interchange of library catalog data. In form it is a short but extremely bushy tree: a collection of some subset of over a hundred possible fields, each of which includes some selection of subfields. The set of possible subfields varies from field to field; typically a field may contain a handful of subfields drawn from a few dozen possible. Some subfields are composed of free text. These texts are usually very short: average length for a title in the Virginia Tech library catalog, for example, is 9.19 words. The longest texts in our catalog are in note fields, with an average length of 13.37 words. Other fields and subfields, while represented as text in the record, are actually fixed items chosen from controlled categories. Most subject descriptors, for instance, are points chosen from the hierarchical *Library of Congress Subject Headings* (Library of Congress, 1988b). Author names are similarly controlled, as are classification numbers, language codes, and standard titles.

For all their domain specificity, MARC records are typical of a wider class of information objects. More and more information retrieval applications now must deal with composite documents: objects where text is important, but is only one part of a complex structure, and where other types of data and indeed the structure itself affect the retrieval task. As authoring systems and standard document interchange formats facilitate more and more highly marked-up documents, the texts that make up individual objects get smaller and smaller, and the structure more and more important. This is nowhere more true than in hypertext systems, where the relevant text objects – the hypertext nodes – may be as small as the titles and notes in a collection of MARC records. Even in purely text-based retrieval systems, attention is again being given to users' needs for answer-passage retrieval: finding useful sentences or paragraphs from a larger text (O'Connor, 1980). Because of these and other applications we consider the problem of retrieval in large databases of small text records to be an important one, and our experience in representing and indexing MARC records to be germane.

Data Model

The overt purpose of a MARC record is to describe a work in a library catalog. Nonetheless, not all fields and subfields of a MARC record describe attributes of the cataloged work. Many are concerned with detailed descriptions of authors, series, publishers – entities in the conceptual world in which the work has its place. This information is often redundant: a person's name, title, birth and death dates and major works may occur dozens of times throughout a collection; a publisher's address thousands of times. The controlled phrases that identify subject headings have even greater redundancy: in the Virginia Tech collection, 90 subject headings occur 1000 times or more, and the most common ("Europe, Eastern") occurs over 23,000 times.

Eliminating Redundancy through Classes of Entities. There are both practical and conceptual reasons to eliminate this redundancy. On the practical side, it takes a fair number of characters to store

Shakespeare, William, 1564-1616

842 times. More importantly, if every work with Shakespeare as author results in an index entry for "William," the already non-trivial problems of finding William Smith by merging the index for "William" with that for "Smith" take on heroic proportions. Similarly, if our task is to find works matching

Words in Subject: number theory

we have a much easier job if we can merge indexes over the class of unique subject headings than over the class of MARC records. There are only 47 unique subject headings in all the Virginia Tech collection that use the word "number," and 285 that use "theory." On the other hand, there are over 6,000 works in the collection that have some subject descriptor with the word "theory" in it. The problems faced here are similar to those in non-normalized relational databases. Although detaching categories of entities from the MARC class does not result in classes that are normalized in the RDBMS sense – in particular, the residue classes of authoring and describing relations are not at all normalized – the impulse and the practical advantages are the same.

On the conceptual side, breaking the classes of authors and subject headings out of the MARC records makes for both a cleaner and a more accurate representation of the bibliographic world. It permits us to treat these classes of entities in ways appropriate to their essential character. A partial match between persons' names, for instance, can be defined in such a way as to take into account the different weight we assign to a match in last, first, and middle names, and to our use of initials for first and middle (but not last!) names. This in turn helps maintainers exercise some control over the data in the classes, identifying possible misspelled subject headings, for instance, or names that may well be less complete versions of other names. Nor is this conceptual advantage without its practical side. Having distinct classes for persons or subjects makes it possible for the user to browse these classes, and thus answer questions that could not be answered by looking at the catalog alone.

The primary class of entities in MARIAN is nonetheless library catalog data: descriptions of books, music, videos, serials, documents, and all the other sorts of works that make up a library collection (Fig. 1). In the current version of MARIAN we recognize four other classes of entities from the bibliographic world. Descriptions of people occur as authors, editors, performers, and so forth, as well as subjects of biographical or critical works. So in the same way do organizations, like *Association for*

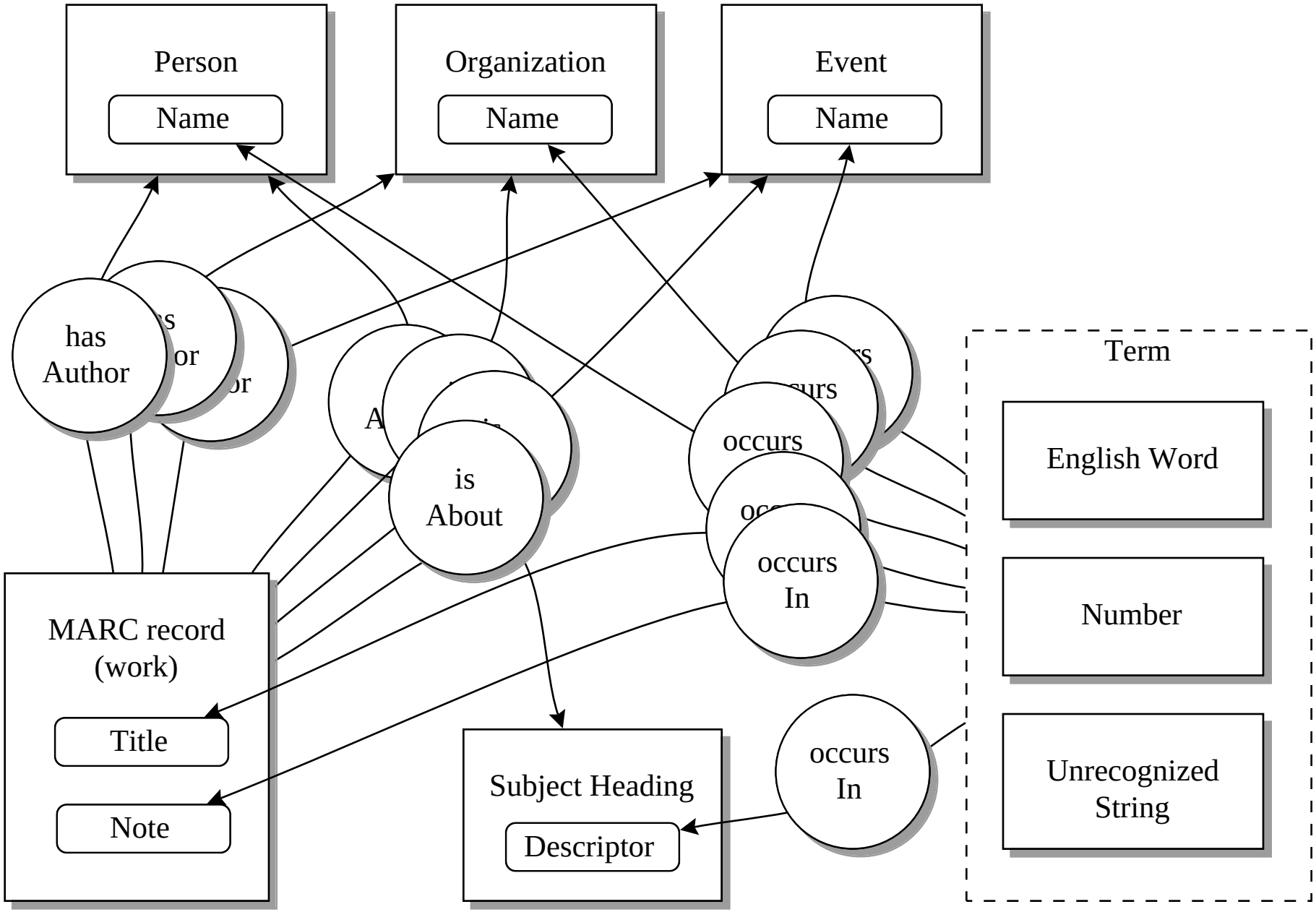


Fig. 1: Entities and links in MARIAN. Composite entities are shown as rectangles, links as lines with circles, and text fields as lozenge shapes.

Computing Machinery – SIGPLAN – Greater Boston Chapter, and events, like *ASIS Midyear Conference '92*. The controlled subject headings form another class. In future versions, we expect to recognize abstract works like *Hamlet* in a category distinct from the catalog records of a particular collection, and to add other controlled classes like dates and call numbers. Finally, the elements out of which free text is composed – words, numbers, and so forth – make up a partitioned class, generally referred to communally as terms.

Links. To connect these entities MARIAN relies on a link construct provided by the underlying database software (Chen, 1992). Links are typed, directed arcs of the sort used by both semantic network and hypermedia systems. MARIAN uses several classes of links, most notably the *hasAuthor* and *isAbout* link classes. The first of these connects a work to one of its authors, and the second to an entity that it can be said to be about. Links can be followed both from source to sink and from sink to source. The relevance feedback engine that we expect to add in the next year will use the *hasAuthor* class in both directions when it moves from a work in which the user has shown interest to all the other works by the same author. Preparing indexes for each direction is straightforward, but the dual directionality presents problems for the efficient storage and retrieval of the links. We discuss the problems and some solutions in the section on Data Processing and Characteristics.

A more interesting problem is posed by the single largest collection of links: the *occursIn* links that connect terms with the texts that contain them. These associations underlie our ability to identify a text object based on a few words, and as such are also the most frequently accessed data in the system. Thus, a paramount concern in storing and indexing them is efficiency of access. Not every mode of access is equally used, however; by far the most common is moving from a particular term to the set of texts in which the term occurs. Accordingly we invert this category of links, and store the information not as links between individual terms and individual texts but as one-way links from a term to a set of texts. The opposite access mode – going from a text to its decomposition as a set of terms – is used in relevance feedback, and can be supported by a similar scheme when we bring feedback on-line.

Information Graph Model: The situation in MARIAN is an example of the information graph model (Fox, Chen, and France, 1991). The information graph model is a set of representation principles designed for unifying divergent data models in a retrieval setting. Without going too far into the underlying formalism, it takes the concepts of node – which in this paper always means entity – and link as a *lingua franca* for relating different organizations in a common framework. In particular, we seek to integrate information retrieval, hypertext, hypermedia, semantic network, and conventional database applications by representing their data in a single information graph of objects, and supporting their operations with an object-oriented DBMS: LEND, our Large External object-oriented Network Database (Chen, 1992). Representing information retrieval data is in part the subject of this paper. By design, an information graph can represent the nodes and links (or relationships) that make up hypertext, hypermedia, and semantic networks. Likewise, our OODBMS can easily represent relational DBMS contents: a relation is really just a collection of tuple objects, each of

which can be thought of as a collection of attribute objects. Collections can be viewed as distinguished subgraphs, or can be explicitly stored by having (numbered) links between collection and collection content objects.

Regarding operations, we support the information graph with several levels of software. At the lowest level, we manage storage and provide indexes through various sorts of minimum perfect hash functions (Fox, Heath, Chen, and Daoud, 1992). The next level manages the underlying graph. Above that layer we consider various views, with special interfaces, such as for information retrieval applications. In particular, the view level supports access to composite objects whose parts are represented in the graph layer as separate but linked objects. LEND also supports composite objects defined as classes of nodes, with their parts hidden by their object definition. The view layer conceals this distinction to users where they do not ned to know about it. MARC records in MARIAN make use of both constructs. The author and subject attributes of a record are represented at the graph layer by links to other entities. The title and note parts are concealed within the MARC node in the graph, where they can be addressed only by class-specific retrieval and display methods.

Composite Objects: The MARC records are an example of a common object in any information-retrieval environment: the composite document or composite object (Fox, 1987). Other examples in the bibliographic world include persons, whose MARC data includes name, numeration, title, life dates, affiliation, major works, and more; publishers; series; and dates. In the abstract, a composite object is a constrained set of attribute-value pairs: a description that associates values with some or all of the distinguishing characteristics for an object of its class. A match between two composite objects, or a directed match between a composite query and a composite object, must therefore be a function that combines the matches of each object or query attribute.

[[Matching functions for composite objects are thus similar to the set-based matching functions used for texts. There are two important differences, however. Matching text objects relies on the semantics of sets, while matching composite objects relies on the semantics of description. When matching text, we want to know how similar one linear combination is to another: how closely the distribution of terms in a text object matches the distribution in another, usually the query. When matching composite objects, we treat the query as a description of the object we are searching for, and measure instead how closely that description fits a particular composite object. In particular, this means that we are less concerned with noise in the case of composite objects. In the case of a text object, we care not only whether the terms in our query are present in the text; we also care how much of the text they account for: are they drowned by a prohibitive amount of noise? In the case of a composite object, we only care whether the parts of the description given in the query are also present in the object. If a MARC record matches:

Author: McCoy **Words in Title:** algebra

it is immaterial to us whether it also has subject headings, notes, or any of the other panoply of bibliographic description.

The second difference is that we cannot scale (weight) the dimensions of composite objects as we can text. The dimensions of a text object – the terms that occur within it – have an implicit information content that can be derived from their global frequency. The dimensions of a composite object have importance that do not appear to be at all

related to their statistics. For instance, we have scaled MARC records in MARIAN so that a match in a note field counts less in the combined match than a match in the title field. This struck us as a reasonable thing to do, not because of any objective properties of the data, but because of a semantic judgement that the title was a more important part of the description of a work than any notes it might have.]]

Design

MARIAN is designed to perform well with vague and incomplete descriptions of works. The system can also function with complete descriptions, and will generally present the described work as the "best match" when the work is in the collection. But it is an important feature of the system that when presented with a request like:

Author: McCoy **Words in Title:** algebra

it will be able to discover Neal McCoy's *Introduction to Modern Algebra.* We believe that many people remember many books using these sorts of cues, and that a library catalog system should be designed to work directly from such a request. This means two things: the system should be able to recognize an author from a last name and a text from a few words, and it should allow the users to combine them in a single request. In particular, users should be able to do this without choosing Boolean operators or handling intermediate result sets.

Matching Functions: MARIAN is designed to provide best matches to partial descriptions, whether the description is of a person's name, a piece of text, or a catalog record. MARIAN is further designed to discover multiple matches to queries, and to present them ranked by goodness of fit. Thus measures of similarity are required at every level and class of object. These measures do not have to work all in the same way, as they would in an n-gram system, but they must all be comparable: the numbers produced by one must agree with those produced by another both in absolute magnitude and in rate of growth. A full discussion of similarity measures and weighting is a topic for another paper. Here we will say only that the category *weight*, together with its own operations and methods of generating weights by comparing objects, forms a foundational class for the system.

Weighted Sets: From the concept of a weight, we define the concepts of ranked and weighted object sets. A ranked set is a set of objects in a well-defined order. In the case of probabilistic retrieval this ranking is based on partial matches, with the best matching documents presented in the top ranks. A weighted set – a set where each element has an associated weight – is a refinement of a ranked set where the ranking is also metric. Weighted sets support the usual set operations: adding an element, testing for inclusion, and so forth. They also support operations based on the members' weights: iterating through the set in order of weight, obtaining a subset of all elements with weight greater than some minimum, and so forth. Some of the normal set operations are changed in weighted sets: the isElement() function, for instance, is Boolean for normal sets but weight-valued for weighted sets.

Ranked and weighted sets are used in several places by MARIAN. The most obvious application involving ranked sets, is to provide ranked results from a retrieval. In that case, the objects are documents (or document surrogates) and the weights are the result of some matching operation. Weighted sets, on the other hand, most often are used to describe the collections of documents that are indexed by different terms. In that case, to process a query that has several terms, one must apply various weighted set combination operations to handle the sets associated with each query term.

Weighted Set Combination. Weighted sets can be combined through the usual operations of union and intersection. Other forms of merging have been advanced in the context of document retrieval systems and pattern recognition. For example, a document can be represented by a weighted set of terms, often referred to as a document vector; in that case we can compare two documents, that is combine two weighted sets, by use of the cosine function. If we use an extended vector representation, whereby a document is represented by a number of subvectors, and want to compare such documents (Fox, 1983), we are in a quandary as to how best to combine the similarities that arise from pairwise combination of subvectors. The Kullback-Leibler minimum cross-entropy function, used in quantum theory and pattern recognition, avoids this problem, but gives non-intuitive results when the set intersection is small (Kapur, 1992). Van Rijsbergen has suggested a measure called information radius which provides yet another figure of merit (Van Rijsbergen, 1979). It is likely that different formulas work better for different classes of objects. We have some tentative evidence that this is in fact the case, which we present in the section on Operation.

Merging algorithms. Whatever formula we choose will be applied within the context of a merging algorithm. And while the formula has some influence on how we index and organize our data (e.g., whether we assign weights during indexing), the algorithm has a great deal of effect on our index structures. All the merging functions mentioned above are summative in nature: they take the form of a (sometimes normalized) sum over the common elements of all the sets being merged. Clearly this summation can be performed only during retrieval. Accordingly, the task during indexing is limited to pre-computing as much of the measure as possible for storage in the indexes for individual elements. However, the very shape of the indexes is determined by our choice of merging algorithm.

We have investigated three merging algorithms during the construction of MARIAN: exhaustive combination, opportunistic combination, and probing. The first creates a complete ranking of all the elements in any of the sets being combined. The second uses “stopping rules” to limit the computation and find a small set of best matches, e.g., one of size 10, or those with weight greater than some high number. The third is relevant in situations where query terms vary widely in frequency of occurrence --- in that case merging of a short and long set is fastest if a quick probe can take place into the long set for each term in the short set.

Exhaustive combination is the simplest, and probably the most commonly used in text retrieval systems. In an efficient implementation of exhaustive search, the search routine creates a table capable of storing any element in the collection of sets, e.g., allocates a table of *n* zeroes when *n* documents are involved (Harman, 1992). It then runs through each set, adding the elements to the table. Where a table cell is occupied, the

element having been a member of some set already covered, the summative formula is applied. When the last set has been completed, the table is scaled as needed, and becomes the resultant weighted set. Various refinements include using some sort of hash table to conserve space, and maintaining the table in weight order to prepare for weight-based set operations.

Opportunistic combination is a variant on this approach where the result set is constructed so that the most highly weighted elements are established early on in the process (Harman, 1992). As long as lazy evaluation of the result set is appropriate, as when only the best matches are needed, it may be possible to avoid establishing the low-weight elements in the set. The weighted sets that occur in document matching are often composed of a few good elements followed by a long tail of relatively poor matches, so this algorithm has the potential for large savings in computation time. It usually involves a noticeable amount of overhead, though, in computing the "stopping rules" that determine whether the top elements are stable.

Probing is most often used in combination with one of the above combination algorithms. A probing search routine makes use of the fact that the sets used in document retrieval vary widely in size. If a search routine has a few small sets and other sets much larger, the most efficient algorithm is often to combine the short sets, then probe the larger sets to determine the overlap. This works well in cases where the weight of set elements is related to the size of the set, as they are in IDF text systems (see below). In that case, the top elements of the result set will be made up of elements from the small sets. Thus the top segment can be established with great confidence, and all that is needed to present it is to determine precise weights for its members. This can be done through probing.

Index Organization for Weighted Sets. To a large extent, preparing the indexes for MARIAN consists of creating and storing various sorts of weighted sets. These pre-computed sets may be optimized for iteration, for merging, or for probing for individual elements. In particular, two sorts of pre-processed sets are used in inverting the *occursIn* link class. We refer to these as posting lists and posting sets.

In a classical inverted file, a posting list is a stored collection of weighted document IDs: pairings of the ID for a document with the weight of its association to the indexing term. Posting lists in MARIAN are similar collections of postings with the added constraint that they are stored in non-increasing order by weight. This means that they are optimized for opportunistic combination. In each posting list, the highest-weighted elements are retrieved first from the collection. Therefore by exploring several posting lists simultaneously, it is possible to establish a frontier beyond which every element has no more than some minimum weight. This is the data from which the opportunistic "stopping rules" work. Posting lists are also effective in exhaustive combination. The MARIAN exhaustive search routine proceeds like an opportunistic search routine, establishing frontiers across the lists to be merged and always adding the highest-weight elements to the holding table first. The table thus constructed is already in weight order except where the sets intersect and weights must be summed. In the common case where merged sets are mostly disjoint, very little extra work must be done to turn the holding table into a new weighted set.

Posting sets are like posting lists except that they are optimized for probing. In the

current implementation, posting lists are stored in MARIAN in order by text ID. Probing is by binary search. Other possibilities include using a minimum perfect hash function to determine the order of the postings, and using some sort of partially-ordered data structure as a heap. Iteration on posting sets is possible, but costly. Finding all the elements in a set above a certain weight, however, is an O(n • log(k)) process, where n is the number of elements in the set and k the size of the resultant subset. Posting sets can thus be effectively used by an opportunistic search routine as well, so long as the search routine stops before much of the set has been explored.

Data Processing and Characteristics

There are three major steps in preparing MARC records for retrieval (Fig. 2). In the first, the records are split into fields and subfields. Members of controlled entity classes – subject headings, persons, organizations, and named events – are checked against existing class databases and added to the database if they have not been seen before. Links between the controlled classes and MARC records are generated. Free text components of the MARC records – titles and notes – and of the new controlled objects – names and descriptors – are collected. The remaining two processing steps deal exclusively with the text collections. In the first, each text object in a collection is analyzed into component terms. In the second, the components are collected to form an index into the collection by term.

MARC Analysis: The first processing step is handled by a single filter driven by a table that associates individual MARC fields and subfields with processing routines. The routines include both pre- and post-processing so that field processing can be sensitive to the subfield processing and vice versa. Setting the main routine up to be driven by a table has proved to be a fortunate decision: as a bunch of computer scientists slowly learn the semantics of the more obscure MARC fields, we have had to change our processing actions many times. Keeping all the MARC analysis in a single table-driven step has made adaptation very easy.

The products of the first step, then, are threefold: updates to the class databases for subjects, persons, organizations, events, and of course MARC objects; links connecting these classes into relationships of authoring or being about; and collections of free text objects. The free text collections are passed to the next step; entity and link classes are ready to be stored in the database. Figure 3 shows the sizes of the entity and link classes produced from the Virginia Tech catalog. Note that the number of links generated is large, larger in total than the total number of objects. This is typical of network-style representation systems. In particular, the number of *hasAuthor* links is much larger than the number of possible authors – persons, organizations, and events – and the number of *isAbout* links is much larger than the number of possible subjects. These two situations actually have quite different semantics, mirrored by the different connectivity of the two link classes.

In the case of *hasAuthor*, only 15% of all MARC records have no author. The author class is small because the number of actual authors is small compared to the number of books written. The subject heading class, on the other hand, is small by

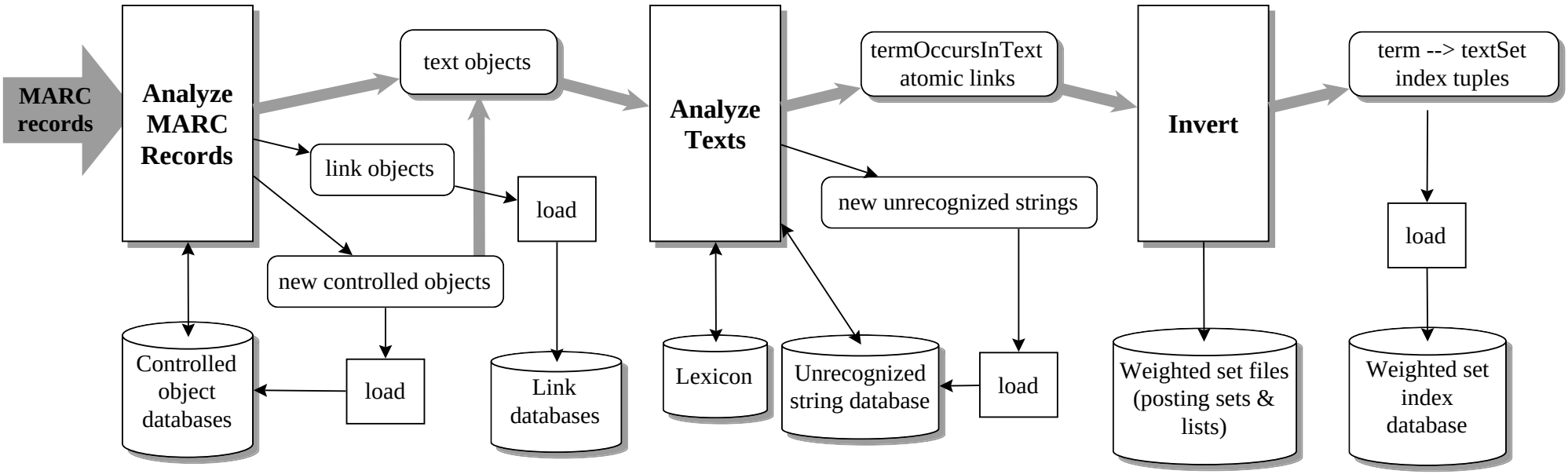


Fig. 2: Major processing steps, intermediate files, and class databases for MARIAN.

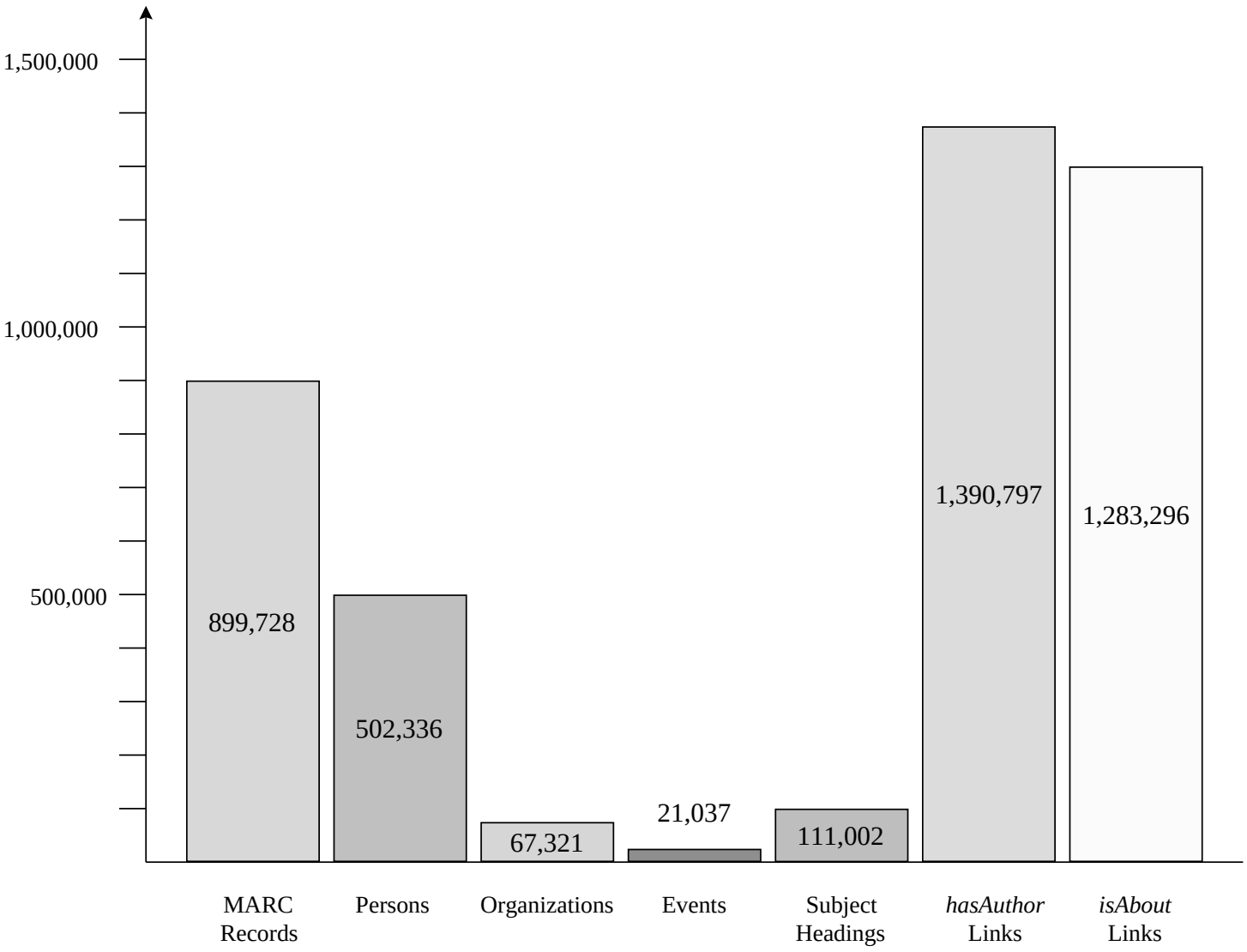


Fig. 3: Number of objects in each of the classes produced by MARC analysis.

design. It would be smaller still if it contained only official Library of Congress subject headings, but we have expanded the category in MARIAN to also include entities and works listed as subjects in the collection records. In this case, though, only half the works in the collection are cataloged with a subject; instead, works tend to have either no subjects or an average of two.

Both link classes have similar patterns of connectivity overall (Fig. 4). In both cases, the number of links attached to the average work is lower than those attached to the average author or subject. Furthermore, in both classes and at each end of the link, most entities with any link have only one. From that point, the number of entities with **k** links drops off as a negative power of **k** in a fashion reminiscent of the Zipf "principle of least effort." The only violation of this pattern is in the number of subject headings assigned to a work, where the drop-off is less steep. This deviation can be explained by standard cataloging practices, which encourage works to be given two or three subject access points when they are given any at all.

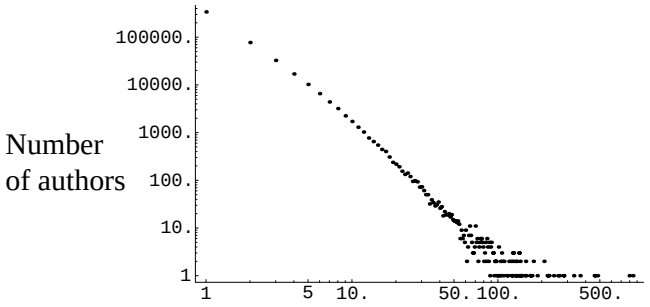
The difference in connectivity comes in the power of **k** on the side of the link connected to the appropriate controlled entity class. From the point in the *hasAuthor* class where fully half the authors are linked to only a single work, the distribution curve drops off from that point at a rate of $1/\mathbf{k}^3$ until it reaches Isaac Asimov with 137 works and Shakespeare with 842. The *isAbout* class, constrained again by its design, has a drop off of only about $1/\mathbf{k}^{1.7}$. This means in practice that while there are few authors linked to more than 20 or 30 works, there are many subjects linked to hundreds or thousands of works.

The difference between classes makes a large difference in retrieval, but no real difference in storing or indexing. The classes are stored like any LEND class: the objects are compressed and packed consecutively into a disk file. Then the classes are indexed like any other LEND link class: a minimum perfect hash function (Fox, Chen, and Heath, 1992; Fox, Chen, Daoud, and Heath, 1991) is built for the IDs of all unique source objects, and one for the IDs of all sink objects. These hash functions allow direct computation of the address and size of an entry in an indirection file specifying the physical address in the disk file of all links with the given source or sink. The important difference for indexing is between the connectivity at the source side – the MARC records – and that at the sink side – the controlled entity class. In both link classes, there is much lower connectivity at the source side than at the sink. In other words, a controlled entity is likely to retrieve a larger set than a MARC object. This is turn implies that when the links are packed into their file, they should be packed so that those with a common sink are placed together and can be retrieved with minimum disk access. This is easily achieved by putting the links in order by sink ID.

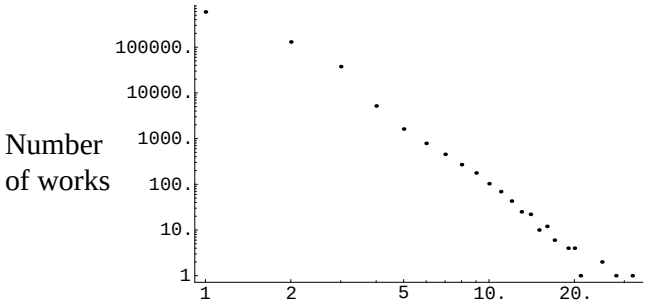
Since both the average and maximum number of links connected to a given source is low, we hypothesize that this unsophisticated solution will carry the day. When link classes have more balanced connectivity, LEND includes clustering algorithms that optimize storage organization so that links that are retrieved together will be as much as possible on the same page (Chen, 1992). These algorithms have some cost, though, so for the moment we are staying with the simple solution.

Text Analysis: The text collections produced by the first step are: from MARC objects, titles and notes; from person, organization, and event objects, names; and from

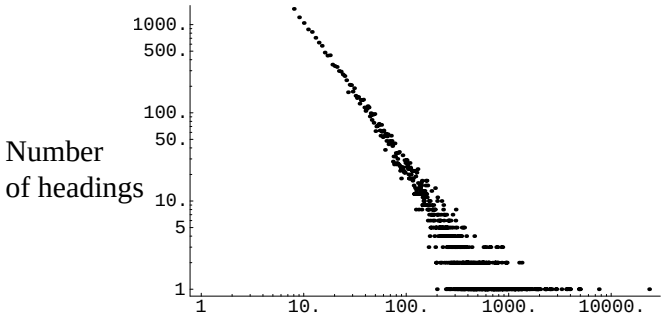
Number of Works per Personal Author



Number of Personal Authors per Work



Number of Works per Subject Heading



Number of Subject Headings per Work

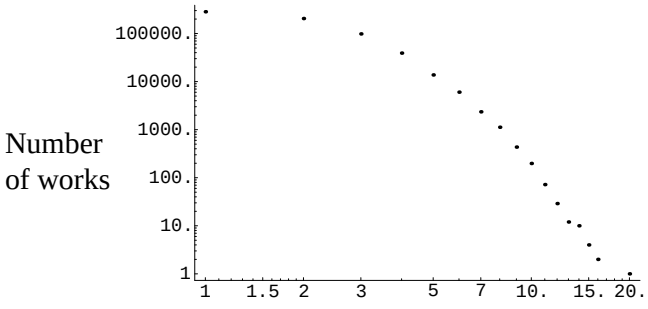


Fig 4: Distribution of links for *isAbout* and *hasPersonalAuthor* classes. The Zipf-like distributions show up clearly in these log-log plots for all except the number of subject headings added to a cataloged work. The shape of the curve there is a result of controlled cataloging practices, which encourage an "optimum" two to three headings per work.

subject headings, controlled descriptors. Each text collection is indexed first by analyzing the texts into the terms that occur in them, then collecting the texts associated with unique terms. These steps are thus respectively local explosion and global re-combination. More specifically, each text is analyzed into a set of atomic *occursIn* links, each connecting a unique term with a unique text; then the entire class of links is restructured into relations between terms and weighted sets of the texts in which each occurs. These two steps are respectively the domain of the text analyzer and the inverter.

The goal of the text analyzer is to identify the components of a piece of text so that they can be matched during retrieval. Simply put, the analyzer breaks piece of text independently into tokens, then assembles the tokens into recognizable terms. In reality, this process is anything but simple.

As each text is accepted by the analyzer, it is broken up into tokens. Tokens are distinguished lexically, as uninterrupted strings of digits, punctuation characters, lower-case letters, mixed-case letters, and so on. This situation is somewhat complicated by the presence of diacritics and European letters in the ANSEL system (ANSI Z39.47), but a program switch allows us to treat non-ASCII characters as either ANSEL or unknown. The token stream is fed to an augmented transition network (ATN) that encodes rules of capitalization, amalgamation, and punctuation. It is the role of the ATN to answer such questions as: Is a period at the end of a letter string likely to signal an abbreviation or an end of sentence? Is a hyphen part of a hyphenated phrase, a number range, or a word broken between syllables at the end of a line? Has a given letter string been capitalized for grammatical reasons, or is it probably a name?

To help it answer these questions and identify terms from the token stream, the ATN draws on recognizers for English words and numbers. The English word recognizer combines a recursive descent parser recognizing common regular affixes and transformations with an 80,000-root lexicon derived from the machine-readable *Collins English Dictionary.* This combination, which we term Lexically Intensive Morphological Analysis (LIMA), allows us to recognize both regular and irregular variant forms of English words while avoiding some of the conflations of unrelated terms that occur with classical stemmers. LIMA also represents a compromise between fast stemmers with little respect for English word formation and true morphological parsers like the well-known KIMMO family (Kostennieme, 1984). The recursive descent component is fast and uses no backtracking, and the lexicon is small enough that most or all of it can be cached in fast memory. We have been using LIMA for several years (France, 1991), and find that it provides good coverage of words in free text. Krovetz (Krovetz, 1993) has reported on a similar but more sophisticated system. His results are also encouraging for this approach.

The MARIAN text analyzer recognizes various forms of numbers, including integers, decimal numbers in various notations, and fractions, as well as numeric codes like ISBNs and social security numbers. Thanks to the completeness of the Collins lexicographers, some names, acronyms, and amalgamated phrases (phrases tied together with hyphens or slashes) are also recognized as words. Other acronyms and amalgamated phrases are identified contextually by the driving ATN. Finally, the inevitable residue of unrecognized strings, including amalgamated phrases not in the lexicon, is assigned to a default category. Amalgamated phrases make up about 55% of the unrecognized string class, and strings involving ANSEL characters another 6%. Of

the 39% remaining, the vast majority are names or non-English words (Fig. 5). This confirms our confidence in the effectiveness of LIMA.

Figure 6 shows how terms are assigned to classes. Figure 7 shows the distribution of recognized terms among classes for three text collections. As could be predicted from former indexing efforts, unrecognized strings always predominate in the set of all unique terms found in all three cases. If we instead count all terms found in a collection, though, recognized English words predominate. In all cases, recognized integers form a small fraction of the total, and the other categories are too small to graph. There are interesting differences between collections: in personal names, for instance, relatively few of the unique terms were recognized as English roots, although even here roots occurred most commonly in text. Among subject headings, roots make up a relatively high proportion of the unique terms. This may be an effect of the care with which the Library of Congress chooses the phrases, but then how are we to explain that the unrecognized strings that do occur account for such a large proportion of the terms found?

The ATN can be switched to handle several different types of text, including English and non-English text using several styles of capitalization: normal English capitalization, text composed exclusively of capital letters, and text with random words capitalized. This last is useful for titles, where capitalization conventions have varied widely – and have been observed irregularly – over the last few decades, and for user input, which runs from no capitalization, through only proper nouns capitalized, to all words capitalized, to no lower case at all. As the ATN produces its links, it associates each one with the frequency of the term in the text. This information is used in the inversion step in calculating the importance of the term in the text. These calculations cannot be made, however, until all texts in a collection have been passed through the analysis step.

Inversion: The inversion step of MARIAN processing involves two independent actions. The individual *occursIn* links produced during text analysis are re-combined to form sets with a single unique term at the source, then stored in forms that favor efficient access of the set of texts at the sink. While this is going on, weights are being computed that measure the strength of association between the terms and texts. These weights are so calculated and normalized that they can be used with a minimum of additional computation during retrieval calculations of how well the text matches a query. The weights are stored with the texts in the resulting inverted sets.

Many weighting schemes are possible – at least as many as the possible similarity functions between two text objects. MARIAN has been built to allow different schemes to be used and tuned; we will describe two in the next section. Most of the schemes make use of global information about the information graph. Such information is typically expensive to gather and sometimes expensive even to compute, so it is important not to realize it during retrieval. Being global, though, it has wide application, and can be re-used during the inversion step. As a result, inversion is a relatively quick process.

Whatever weighting scheme is chosen, inversion reduces a collection of *occursIn* links to a collection of weighted sets. These sets are not all equal. Among the 500,180 unique words found among the Virginia Tech MARC records, about 2/3 (337,407) occur only once. Most of the remaining occur only a few times each, and only 58 occur more

J.Z.	Meuharim
Bockle	Monacensia
Burobauten	ungefahr
Lubecker	erzahlerischen
circonstanciee	B.II
flotenspiel	L’Orfeverie
Rocenka	L’A.S.B.L
Chabrovice	Binimat
Beliak	Winzentsen
Bunrigakubu	Schwingungstechnik
Dugdale	Pribil
R.J.F.	Jamvikt
Tsadasy	Macrophylla
Lunsford	Cebysev
Ajuria	Ulunian
Muenscher	d’aerodynamique
Degommage	Modjokerto
D’Ap	provlemata
SARON	geraldika
Jicoteitat	aromnaia
T.13	inozemtsev

Fig. 5: Random sample of unrecognized strings, excluding amalgamated phrases and terms using ANSEL characters.

Class	Example in Text	Recognized As Object
word: root	man	#ROOT:52080#
: regular inflection	manliness	#ROOT:52080# + [ly, ness]
: irregular form	men	#ROOT:520-80# + [irrPlural]
number: integer	1952	#INTEGER:1952#
: decimal	19.52	#REAL:<19.52># *
: fraction	19/52	#RATIONAL:<19/52># *
code number	077-24-3593	#CODE_NUM:2347890#
amalgamated phrase	and/or	#ROOT:20406#
	object-oriented	#UNREC_STR:2347891#
unrecognized string	Asimov	#UNREC_STR:4261760#

* NOTE: The instance ID of a real or rational number is a direct encoding of the number.

Fig 6: Example text components and how they are categorized during text analysis. Objects are represented by ID pairs: class ID (here written as a symbolic constant) and instance ID within that class.

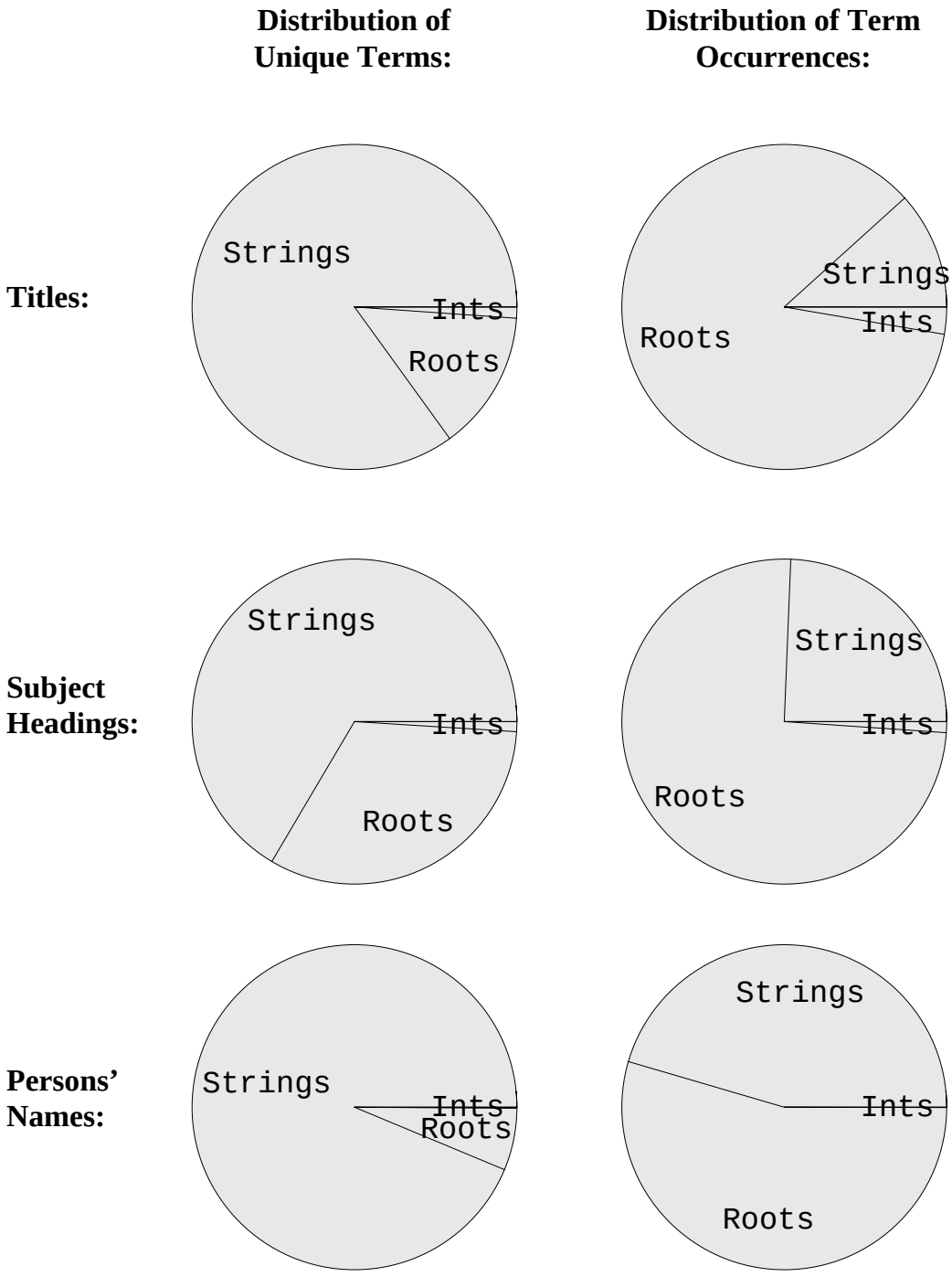


Fig 7: Distribution of terms by type for three categories of data, showing relative numbers of integers, English roots, and unrecognized strings. Other categories were too small to be significant.

than 20,000 times each. This is in accordance with Zipf's well-known Law (Zipf, 1949; Miller, 1957), which predicts that in any sufficiently large body of text the rank of a term in descending frequency order is inversely proportional to its frequency. As can be seen from Figure 8, this relationship holds in all the text collections derived from the Virginia Tech MARC records, even though the high-frequency terms vary widely between collections. In the title collection, for example, function words occupy the top positions, while in the collection of persons' names these positions are filled by initials and common first names (Fig. 9). Zipf's Law further applies whether we are counting number of occurrences of a term, as we have above, or number of documents that contain a term.

We exploit Zipf's Law in MARIAN in several ways. In constructing a text index, we take advantage of the effect by using different sorts of weighted sets for different portions of the Zipf curve. Specifically, we divide the curve into three regions: the steep region in the top ranks, where each term occurs in many texts; the middle region where each term occurs in only a few texts; and the long tail, where each term occurs in only a single text.

The terms that occur in only a single text each are converted to singleton weighted sets. Where a term occurs in only a few texts, the texts are sorted into order by decreasing weight and stored as a posting list. Where a term occurs in many texts, the texts are stored as posting sets, in order by ID. In the usual flow of processing, this last involves little or no sorting. Text IDs are usually assigned consecutively to new texts, which when fed through the text analysis filter produce a stream of *occursIn* links in text ID order. The sub-sequence of the stream of links with a common term will thus also be in text ID order, ready for storing as a posting set. This also works for composite objects with multiple text fields. The stream of titles from the MARC analyzer, for instance, take the form of a sequence of associations between MARC IDs – assigned consecutively by default – and title texts. When the texts are analyzed, the resulting file of links is in order by MARC ID. Thus the only inverted sets that need to be sorted are the posting lists, which by selection are always relatively short.

Posting lists and posting sets are stored directly on disk by the inversion program. When common information is removed from the postings it is possible to pack these files very densely. Offsets in the packed files and the lengths of lists or sets are stored in a master index of inverted sets. Singleton sets are stored directly into the master index with the weight and ID of the single text packed to fit the same size field as the offset and length for the larger sets. This master inverted index has the form of a link class between term objects and weighted set objects, where the weighted sets are tagged to determine their interpretation as singleton sets, posting set, or posting lists. Unlike our other classes of links, this class need only be indexed on the term side, since no retrieval operation takes an entire weighted set to the term associated with it.

Figure 10 shows the relative sizes of files produced during processing of the MARC title field collection. Beginning from a file of close to 900,000 texts in 64.5 Megabytes, the text analyzer produces a file of 7.5 million *occursIn* link objects in 216 Mb. By comparison, recall that the same body of MARC records produced only about 1.4 million *hasAuthor* links and 1.3 million *isAbout* links. To complete the picture, MARC note texts produced another 4.5 million links, and texts from the controlled entity classes together produced about 2.1 million. If stored in this form, the *occursIn* links would

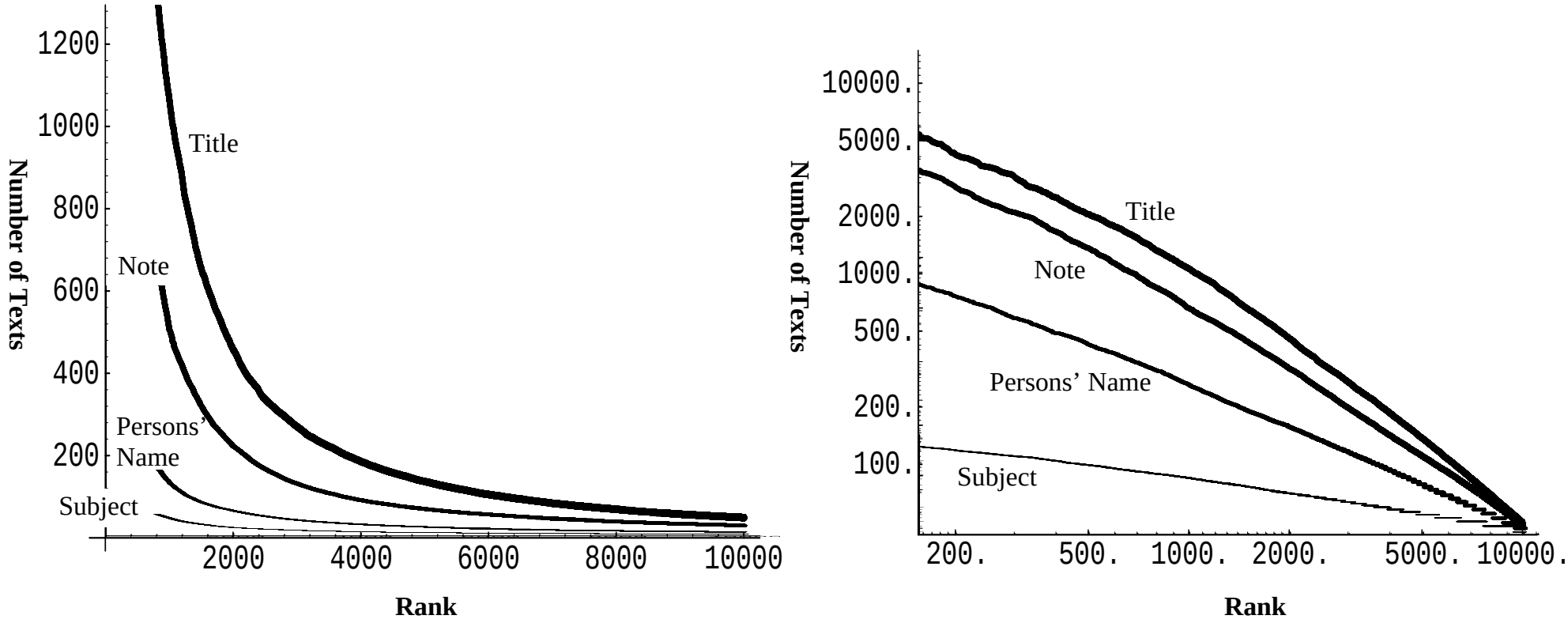


Fig. 8: Observed rank/frequency curves for text in the title and note fields of Virginia Tech MARC records, the names of unique persons, and for unique subject descriptors. The left-hand graph, with linear axes, shows the characteristic shape for the central portion of the data set. When the entire data set is graphed, very little of the curves can be seen, as they are forced by their extended tails to hug the axes. The right-hand graph shows the same data in a log-log plot. In this format, a distribution that followed Zipf's Law exactly would be represented by a straight line.

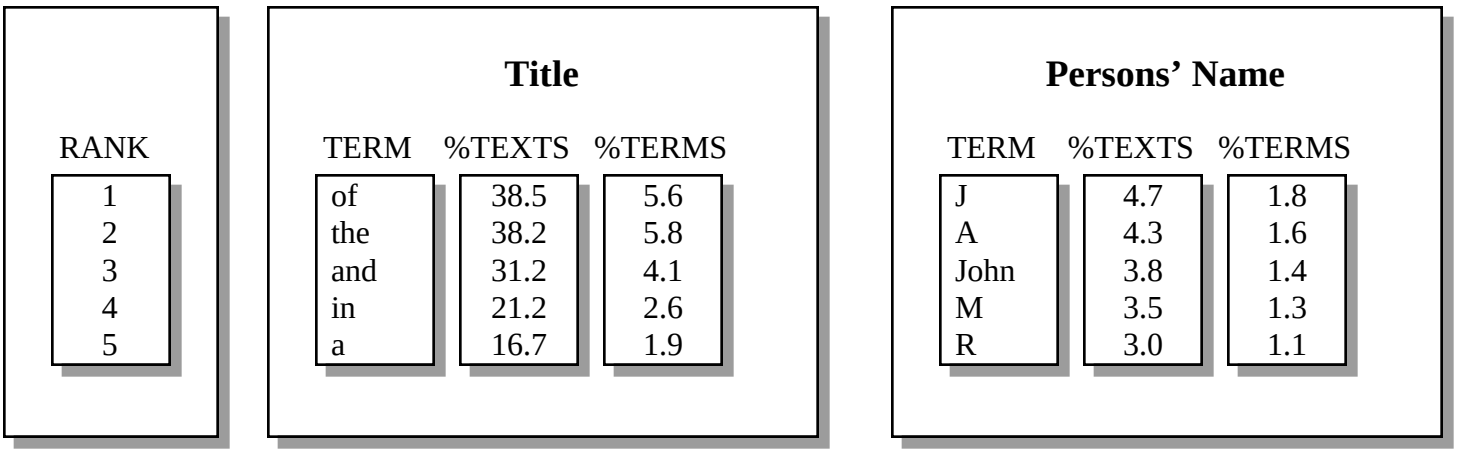


Fig. 9: Top-ranked five terms in *MARC Title* and *Persons' Name* text collections, together with the percentage of texts containing the term and the percentage of all the term instances in the collection that it accounts for.

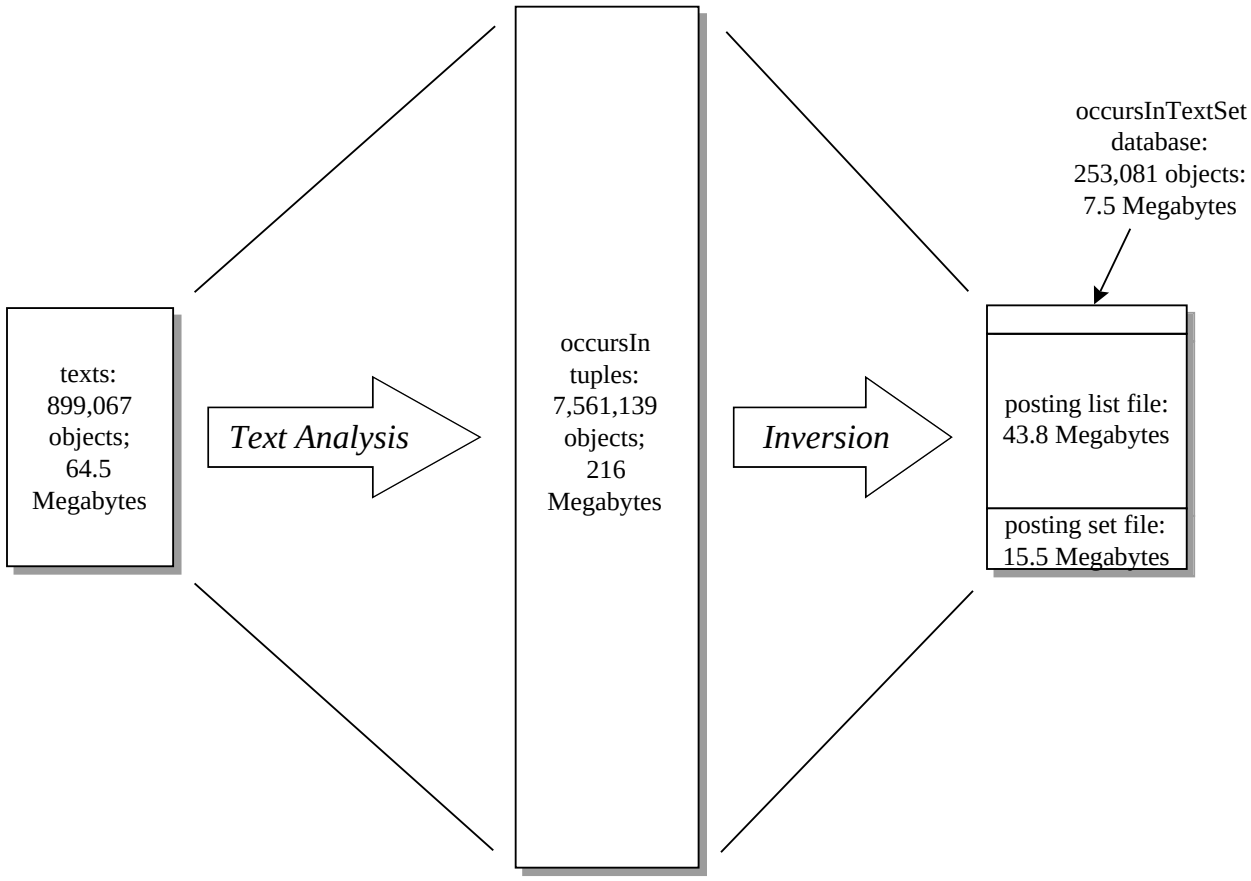


Fig. 10: Expansion and contraction of data during text indexing. Vertical size of boxes is proportional to size in Megabytes. Figures are for title collection, but are typical.

make up almost three quarters of all the objects in the information graph, and account for as much storage as the complete unprocessed MARC data. After inversion, in contrast, they take up barely more space than the text that produced them, distributed among less than 0.8 million objects. In fact, only a small fraction of the storage (the inverted link objects – 7.5 Mb in the case of the title collection) must actually be indexed and stored as a database: the other 59.3 Mb consists of packed weighted set objects.

Stop Lists: A fourth region of the Zipf curve should also be possible. Zipf's Law predicts that the most frequent few terms each occur a huge number of times. For instance, among terms occurring in titles in the collection, the top-ranking 5 terms account for 20% of all the tokens in the collection, and the top-ranking word occurs in two out of every five titles (Fig. 9). These high-frequency terms have correspondingly low information content, as they do not serve to distinguish one text from another as well as do lower-frequency terms. This effect is often exploited in information retrieval systems by putting most or all of the high-frequency words on a "stop list" of unindexed terms. Other candidates for the stop list include low-frequency prepositions and connectives, numbers, and words like "not" whose effect on the meaning of a text is considered problematic.

In the case where texts are large and include many different terms, the use of a stop list appears to have little effect on retrieval effectiveness. Moreover, a stop list of high-frequency words has a salutary effect on system performance. After all, the five words in Figure 9 together account for 1/5 of all the term-text associations in the collection. Ignoring that fraction both shrinks the total size of the generated indexes and reduces the largest set of texts associated with a term to a size that can be handled with ease.

Stop lists are not without cost, however. Although the terms on stop lists are selected to have very low information content, they are not totally without differentiating ability. This is well known in the OPAC community, and a few examples will demonstrate why. First, consider the search request:

Words in Title: to be or not to be.

In many retrieval systems, both statistical and Boolean, and in many keyword-based OPACs, this query will draw a complete blank: all the words used are on the stop list. In point of fact, though, there are 11 books in the Virginia Tech catalog with the phrase "to be or not to be" in the title. Next, consider the request:

Words in Title: On the beach.

The non-stop word "beach" occurs in 388 titles in the Virginia Tech catalog. Only twenty works, however, also contain the words "on" and "the," and only Nevil Shute's novel and the film made from it match the request exactly (Fig. 11).

Classical OPACs approach this difficulty by constructing "title keys" as surrogates for title indexing. Typically, the key is built out of the first few characters from the first few words of the title, skipping certain leading words like "the" and "a." The disadvantage of title key indexes is that they are only effective when the user can remember and enter the title exactly. If the user gets word order wrong, misremembers prepositions, or drops significant words from the beginning of a title, her query will no longer match the key in the index.

In the case of collections of short texts, stop lists become even more problematic. The function of the word "of" in differentiating the title *Theory of Numbers* from *Number Theory* cannot be predicted from its information content in the language as a whole. To

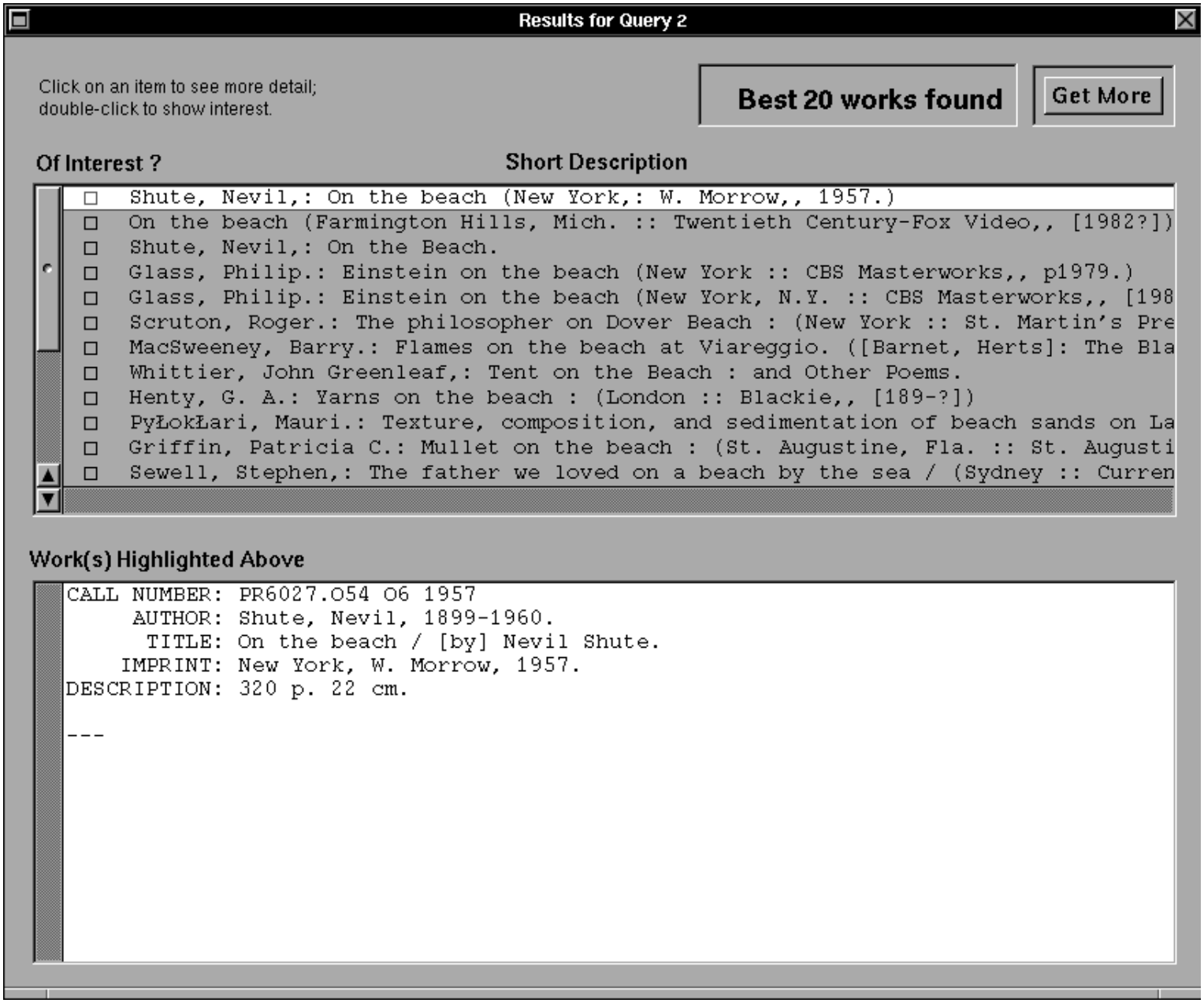


Fig. 11: Results from MARIAN for the query "**Words in Title:** On the beach."

clarify this, we need to differentiate the information carried by a term *in* the text from the information the term carries *about* the text. The first is a measure of how much the term contributes to the content of the text; the second a measure of how well it differentiates the text from others like it. All terms carry information *in* the text, how much information being a function of their frequency in the language as a whole. Not all terms in a text carry information *about* the text, however. It is only when a term frequency differs from its value in the language as a whole that it carries information about the text. How much information it carries is a function of the deviation of its frequency in the text from the average or expected value of a text in the collection. The more wildly it differs, the more information.

In longer texts, high-frequency terms – terms low in information in the text – tend to approach their statistically expected value and thus contribute little to our information about the text. When a text contains only a handful of words, however, any non-zero term frequency will differ wildly from its expected value. This is because, except possibly for the top-ranked word or two, *all* expected frequencies are less than 0.1. To put it another way, if an instance of "of" is missing in a paragraph, or even if all instances of "of" are missing, we can still make perfectly good sense of the paragraph. If the middle word is missing from the phrase "theory — numbers," we don't know if the phrase is "theory of numbers," "theory and numbers," or "theory without numbers." In small texts, every term must be regarded as significant.

Operation

Searching in a collection of complex objects presents its own problems, both in the search engine and in the usability of the system. The user must be able to easily create queries that refer to composite objects in such a way that the underlying system can recognize what sort(s) of object(s) will satisfy the user's request and what parts of the query apply to what parts of those objects. Doing this in a general way is difficult. Moreover, MARIAN is intended to serve as a production OPAC. Library patrons do not typically use OPACs often enough to become proficient, and have little patience with figuring out the details of a complex system. Accordingly, MARIAN query objects are not general, but are specialized to the sort of object being searched.

When a user submits a query for a MARC object (Fig. 12), she does so by filling in a set of fields with text. MARIAN then converts the query form into an object in the underlying data model. Text fields are passed through the same ATN parser and LIMA recognizer as the original data. The resulting query object is collected by a composite object search routine specialized to MARC objects.

In the case of the query illustrated:

Words in Title: number theory **Words in Subject:** number theory

the search routine is looking for MARC records whose titles match the text object "number theory" and which are connected by an *isAbout* link to a subject object whose descriptor matches the same text object. Whether the search routine chooses to merge the weighted set resulting from the title match with the weighted set resulting from the linked object match, or chooses to perform one match and then probe the other weighted set for values depends on the relative sizes of the two sets.

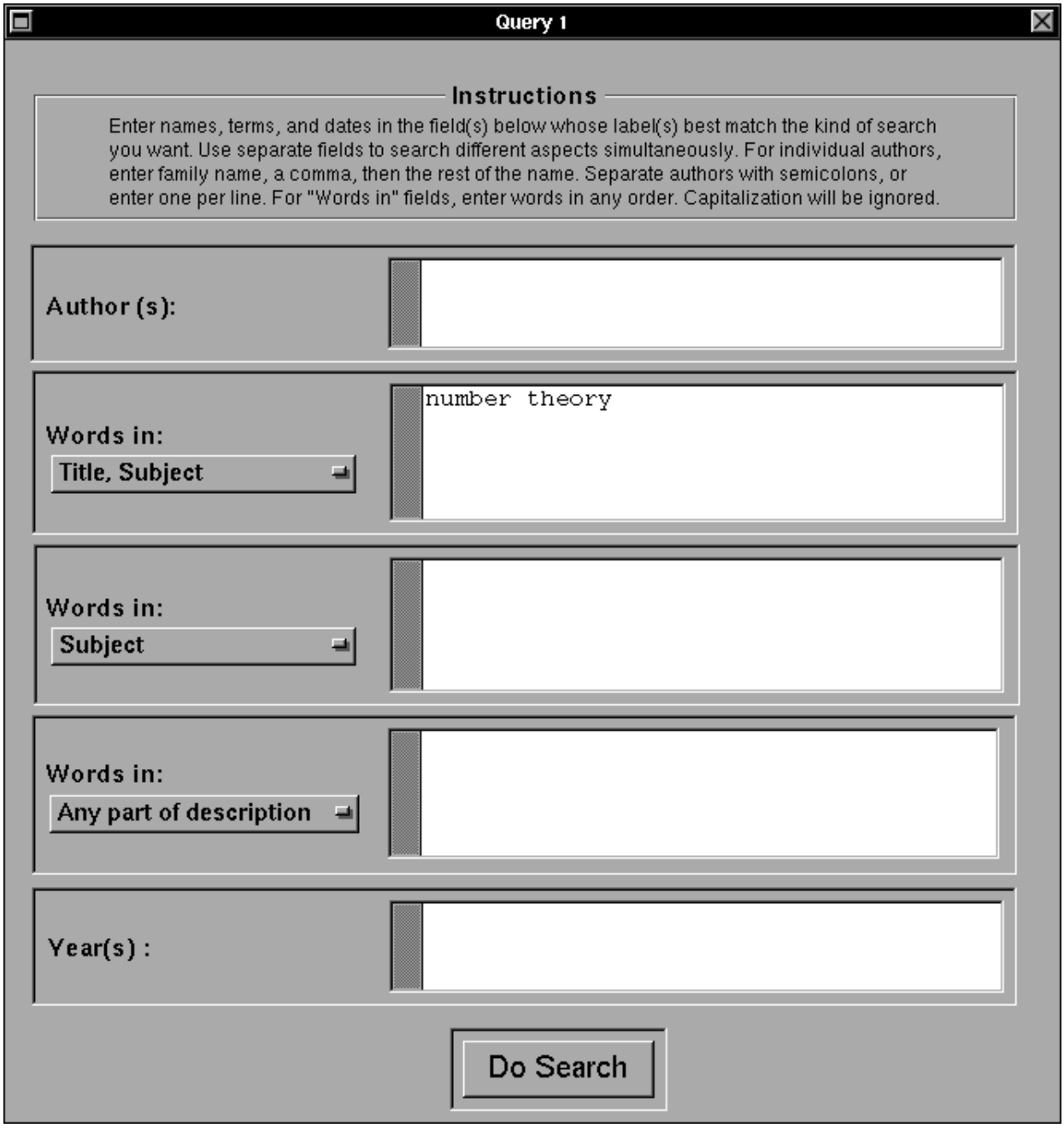


Fig. 12: A multi-part MARIAN query. Equivalent queries could be constructed using this form in a number of ways, most simply by setting the coverage of the text field to "Title" and copying the text to the field below.

In either case, the linked object set will be constructed in the same way. The text object corresponding to "number theory" (two object IDs of class *root* with equal relative weights) is sent to a text search routine with instructions to search against the descriptor field of the *subject* class. The text search routine returns a weighted set composed of the subject "Number theory" with a perfect match value, followed by (in some order) other subjects that contain the two words with some "noise" and subjects that contain only one of the words, with first less and then more noise. The first group includes subjects like "Algebraic number theory" and "Number theory – Congresses;" while the second includes "Surreal numbers," "Million – the number," and eventually the music subject called simply "Theory." How these two groups are ranked depends on the weighting function used for text objects, as discussed below. From the returned weighted set, the link search routine constructs a weighted set of MARC objects with at least one *isAbout* link to at least one *subject* object in the set.

The composite object search routine for MARCs has sent a concurrent request to a text search routine to perform the same text search against the title field of the MARC class. This produces a similar, but larger weighted set of MARC objects with the text in their title. This set is larger than that for the subject descriptor, and includes texts with larger amounts of noise, but is otherwise similar. The MARC search routine then chooses an algorithm based on the sizes of the two sets and merges them to form a single set of MARC objects for display to the user. This set (Fig. 13) is presented to the user in order of decreasing weight. The top of the order includes works with "number theory" in both title and subject, as requested, and with minimal noise. Further down in the set, works occur with greater noise, as do works that lack one of the words in title or subject. Again, how these are ranked depends on the specifics of the weighting functions used.

Similarity Functions: As mentioned above, MARIAN is designed so that different weighting functions can be used during similarity computations. There are three places where similarity functions are needed: between texts (sets of terms), between linked objects (objects tied together by a link or series of links), and between composite objects. As yet we have no conclusion on similarity functions for composite objects. We have tried vector cosine, which has been suggested both by Fox (Fox, 1983) and by Tuevo Kohonen (Kohonen, 1977). We have also tried a simple weighted average of the component similarities. Both work well. In fact, the two agree often enough, and disagree in such obscure cases, that we have not pursued the question further.

Linked Objects: It is the nature of links to be binary in weight: two objects either are linked or are not. This is particularly true of the *hasAuthor* and *isAbout* links: every book authored by a person is equally connected to that person; every subject of a book is (at least, in the abstraction of a catalog record) equally the subject. The similarities between link *classes*, however, are not binary, but are based on the semantics of the two classes. A request for:

Author: Lillian Hoban

certainly matches best the books that Hoban actually wrote. But books she illustrated or edited are not irrelevant to the query. We express this in the information graph model by saying that a *hasAuthor* link matches another *hasAuthor* link perfectly, matches a *hasIllustrator* or *hasEditor* link less well, and matches other links not at all.

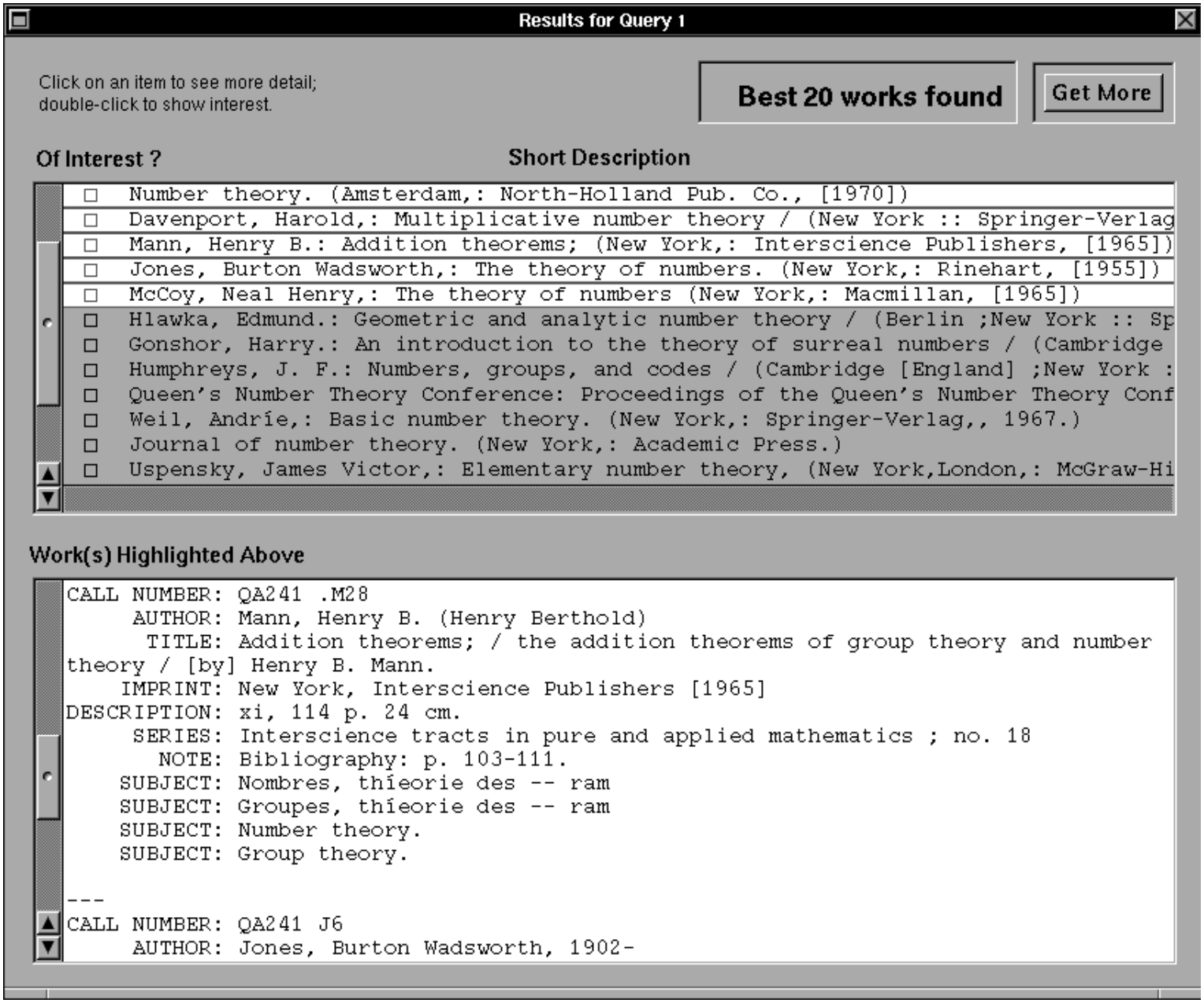


Fig. 13: Results of the query in Figure 11.

Our current matching function for linked objects takes the product of the link match value with the object match value. In other words, if we are matching an object one link away from our target object, we multiply how well the linked object matches our query with how well the link matches the relationship in the query. This is a recursive measure: if an object is two links away, and the intermediate object is a perfect match (or more commonly is not specified in the query), then the match of the complete path is the product of the second link with the combined match of the initial link and object. More generally, if a path is defined as a linear sequence of object and links with the last link dangling, the formula for adding another object and dangling link to the end of the path is the product of how well the path matches, how well the new object matches, and how well the new link matches. This allows us to calculate closeness of fit for linear sequences of links. Situations where two or more different links converge on the same object are treated as composite objects; that is, how closely the object matches a query is a summative function of how well the one path matches the corresponding path in the query with how well the other matches its corresponding path.

One situation that is not covered by the cases above turns out to be fairly common in MARIAN. Let us say that we are searching for

Author: Lillian Hoban

There are many authors in the Virginia Tech collection that match this request at a non-zero level: Lillian Hoban herself matches it perfectly, but any other Lillian or Hoban match it to a lesser extent. In particular, her frequent co-author Russell Hoban counts as a partial match. Thus a book by the two Hobans has two incoming paths with positive matches, each of which satisfies the query.

In its first implementation, MARIAN treated this situation under the rubric sketched above for composite objects, and set the overall match of such an object at the (weighted) sum of the two path similarities. This produced very odd results. In particular, books written by Russell and illustrated by Lillian got higher marks than those that Lillian had written herself. Similarly, a search on

Words in Subject: model theory

found works with many *isAbout* links to different sorts of modeling; these got higher ranks than those with only a single link to the subject "model theory." Our response to this problem was to replace the summative function with a function that simply returned the maximum of the possible matching values.

There is semantic justification for this. A link can be interpreted as an attribute of the linked object. This is clearly the case with our *hasAuthor* and *isAbout* links, and it is equally true of their inverses *isAuthorOf* and *describes.* For links that can be so interpreted, if an object is linked by the same class of link to two objects of a common class – if it has two acceptable values for a given attribute – it is reasonable to claim that a query object with a single value for that attribute can match either, but not both. If it has non-zero similarity to each, we can optimistically choose the best. But we cannot somehow claim extra points for matching both. Similarly, in the case of a two-value query matching a pair of linked objects – say, matching

Author: Francis **Author:** Kucera

against W. Nelson Francis and Henry Kuˇcera, we will only believe that the matching formula is well-behaved if it matches "Francis" against Francis, and "Kucera" against Kuˇcera; which is, in fact, the maximal match among the four possible.

In the end, though, the real justification for using the maximal match rather than a sum or average is operational: it is universally agreed by all users of MARIAN that using a summative measure in this case promotes the wrong books.

Text: Following the "vector space model" (Salton and McGill, 1983), we represent texts as linear combinations of terms. During text analysis, each text is mapped to a set of *occursIn* links from all unique terms that occur within it. As part of the mapping, each link is assigned a local weight based on the proportion of the text accounted for by the term. In the inversion phase, each term is assigned a global weight based on its frequency within the whole class of texts. Local weights allow us to match the distribution of terms in a query to that in a candidate document: to discover how much of the query the document covers, and how much of the document is "noise" not accounted for by the query. Global weights express the information-carrying power of the terms in the context of the text collection. The match between a query and a text can be expressed as a function of the two.

Several such functions are available, and MARIAN has been designed so that we can easily change the function used. To date we have tried a cosine similarity function with TF and IDF weights, and an information-theoretic measure with initial weights derived from the TF and IDF values. Somewhat to our surprise, the ad hoc information-theoretic function performed better at ranking retrieved documents.

Many functions pass under the rubric of vector cosine and weighting using TF and IDF. For the MARIAN text similarity function we tried to use a theoretically defensible version that is similar to one use with extended Boolean retrieval (Fox, 1983). In particular, we used the TF formula:

$$TF(i, j) = 0.5 + 0.5 \bullet Ct_{ij} / Tot_j$$

where Ct_{ij} is the number of times term i occurs in text j and $Tot_j = \Sigma\ Ct_{ij}$ is the total number of terms in the text. This formula has the property that it varies only through half its magnitude, so that the difference between the least possible TF value and the highest possible is less than the difference between the least possible and zero.

The IDF (Inverse Document Frequency) component of our weighting scheme conditions terms with lower information content to have smaller impact on text similarity. In effect, we weight terms by their information carrying ability in the context of the text collection as a whole. This purpose is shown by the classical IDF formula

$$IDF(i) = \log(N / n_i)$$

where N is the number of texts in the collection and n_i the number of texts containing term i. This is equivalent modulo the unit of measurement to the instantaneous entropy of the term

$$S_i = - \log(p_i)$$

which measures the amount of information term i carries in the language.

Our similarity function uses both of these measures. First we define the weighted inner product of two vectors $<a_i>$ and $<b_i>$ as the real number produced by the sum:

$$(<a_i>, <b_i>) = \Sigma\ (IDF(i) \bullet a_i \bullet b_i)$$

This measure differs from the standard Cartesian inner product by its use of scaling constants – here, the IDF values – once in each of the components of the sum. It is nonetheless a valid inner product function, and satisfies the appropriate axioms. The

norm (or vector length) of a vector is the square root of the inner product of the vector with itself:

$$|| <a_i> || = (<a_i>, <a_i>)^{1/2}$$

If we regard the vector for text j to be <TF(i, j)> and that of the query q to be <TF(i, q)>, then the similarity between q and j is:

$$Sim(q, j) = \frac{(q, j)}{|| q || \bullet || j ||}$$

As an alternative to vector cosine, we have also configured MARIAN using a simpler, if rather ad hoc function. In this configuration, the similarity function is simply the weighted inner product described above, without the norms, and with a slightly different TF function:

$$ITF(i, j) = 1.0 - \log_s(Tot_j / Ct_{ij})$$

where $s = (Tot_{max})^2$, the maximum number of terms in any text in the collection squared, is a normalizing metric chosen to make sure that ITF(i, j) varies between 0.5 and 1.0 as does the TF function above. The term $\log_s(Tot_j / Ct_{ij}) = -\log_s(Ct_{ij} / Tot_j)$ is the instantaneous entropy of the term in the text, modulo the unit s. It varies from 0.0 when $Ct_{ij} = Tot_j$ to a minimum of 0.5 when term i occurs only a single time in text j and j is a longest text in the collection.

Since the ITF measure varies from 0.5 to 1.0 and the IDF measure used in the weighted inner product is normalized to vary from 0.0 to 1.0, the maximum possible value for any component of the similarity measure is the relevant query weight. We thus scale the sum by the sum of the query weights to obtain a figure of merit for the match.

Experiment

To test the appropriateness of the weighting schemes used during indexing of the catalog data, we have performed two types of studies. The first was an informal solicitation of comments from users testing our system. From a variety of discussions, it seemed that users were surprised by the rankings that resulted from using our relatively standard weighting involving TF and IDF, with cosine correlation. On the other hand, they seemed to like the results when our ad hoc function was used.

Therefore, we decided to undertake a second study: a simple experiment to compare the effectiveness for the two weighting schemes. For this we desired a representative set of user information needs. Having recently completed the analysis of a large scale library catalog experiment in which we solicited information needs from patrons of our library (Fox and Wilson, 1991; Daoud, 1993), we elected to use queries taken from that experimental effort.

Our design was relatively simple. We worked with 16 students in the Fall 1993 class of CS5604, Information Storage & Retrieval, asking each to make relevance judgments related to five queries. From our earlier experiment, we selected 16 queries, and then randomly assigned the results for five queries to each student. Thus, we could determine relevance as a function of the judgments of five different “experts” for each of 16 queries.

For each query we retrieved the top 20 documents using MARIAN ranking, on two versions of our catalog database of approximately 900,000 records. As discussed in the previous section, one version uses "Cosine - TF*IDF" and the other "Weighted Inner Product - Ad Hoc." For each query we constructed the union of the documents that were found by the searches for each version, randomized the results, obtained relevance judgments on a Likert scale from five students, and then used averaging to develop an overall binary relevance score.

Figure 14 shows the recall-precision results using the SMART evaluation package for the two weighting methods studied. Other measures of overall performance gave similar conclusions regarding the superiority of the ad hoc approach. We plan on repeating this study with other weighting schemes and to carry out statistical and document preference relation comparisons on the results in the future.

Conclusion

This paper deals with a variety of issues relating to the indexing of small text records in preparation for ranked retrieval. While we focus on one particular OPAC system, MARIAN, we believe our results are of interest as other OPACs that do or will support ranked retrieval. Further, we believe our approach and findings are of interest to developers of ranked retrieval systems that at some point must deal with small text records, such as would be found in a highly tagged SGML database, or in a collection of small hypertext nodes.

MARIAN is an object-oriented system, from its design, to its programming, to its use of our LEND object-oriented DBMS. Accordingly, we have re-formulated the data modeling, indexing, weighting, and ranking problems from an object-oriented perspective, and make use of the abstract concept of weighted sets to optimize our data structures and retrieval processing. By applying our information graph model, we have been able to more carefully describe our indexing process and the resulting representation, which is designed to facilitate partial, flexible matching at all levels. We believe that the result is not only good system performance for retrieval, and relatively compact storage of our collection data, but more careful and accurate identification and utilization of the "objects" that are of interest to our users (and to librarians). By studying the resulting classes of objects we were led to a new ad hoc weighting scheme that seems to give better retrieval performance than another, relatively standard scheme, that we considered initially. Presumably, this occurs because we are beginning to learn where "noise" in the documents can be ignored -- that is, where satisfying a query is more important than matching it.

We have found that in the case of small texts, every word is important, and so do not make use of stop lists. While this can cause problems with response time in a search system based on posting lists, using a mixed set of search strategies usually provides acceptable performance. This is sometimes also true of opportunistic search methods. Preliminary results indicate that these do not work well on collections of small texts, but they may still be of merit for searching collections of linked or composite objects. We plan on continuing to study the interplay between effectiveness, search time, and indexing methods in this very interesting domain of online catalogs.

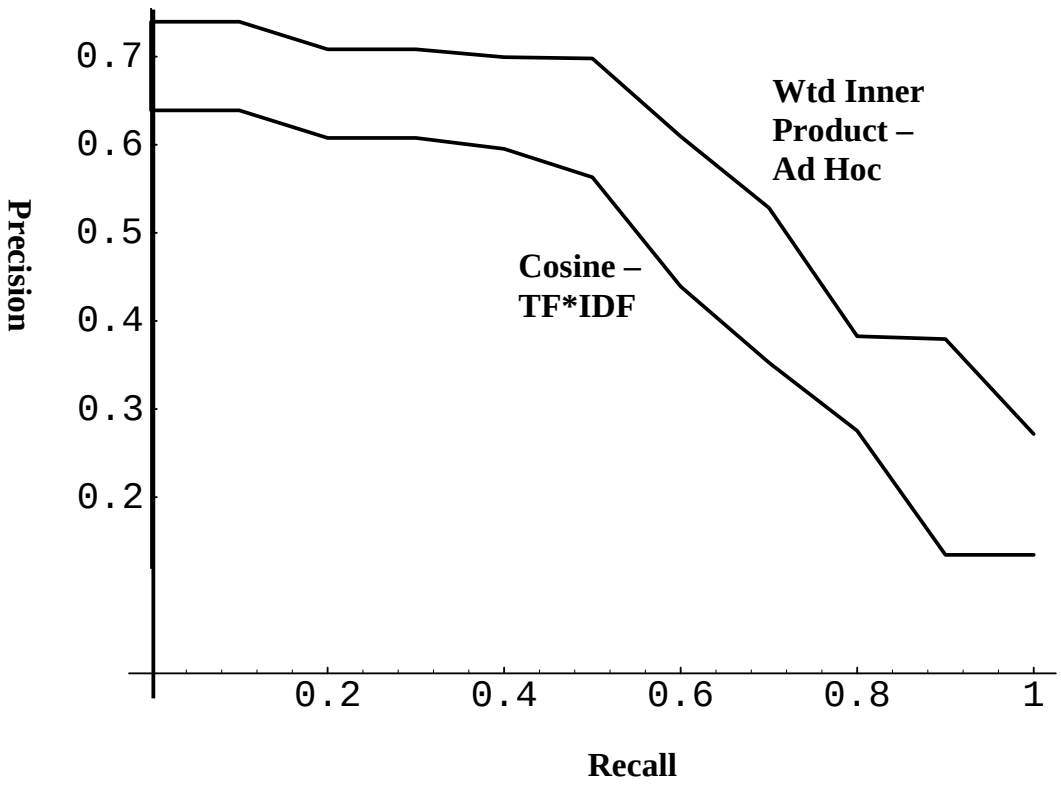


Fig. 14: 11-point recall-precision curves for MARIAN running two different similarity functions: vector cosine with TF*IDF weights and weighted inner product with ad hoc information-theoretic weights.

Acknowledgements

We acknowledge the assistance of all who have worked on the REVTOLC study (especially Amjad Daoud and Linda Wilson), which was funded in part by OCLC, and which helped us obtain support from the Computing Center to begin the MARIAN project. Ben E. Cline of the Computing Center has worked on every phase of the MARIAN project, and made many values contributions to both its design and operation. He is also the proud author of the "On the beach" example. Steve Teske has contributed extensively to the data processing described in this paper; in particular, he is the author of the MARC analyzer. The LEND database over which MARIAN is built is the brainchild of Qi-Fan Chen. Sangita Betrabet has maintained the code and added object classes to adapt it to MARIAN; more recently her duties have been taken over by Liya Huang. Mr. Teske and Ms. Huang also helped prepare data for the experiment described above. The actual subjects, whom we thank one and all, came from Dr. Fox's CS5604 class. Recall-precision values were calculated by the teaching assistant for that class, Joseph Shaw. Finally, MARIAN's excellent user interface was crafted by Eskindar Sahle, with initial guidance by Tim Rhodes and usability testing by Lucy Terry Nowell. We especially thank the Computing Center for their support of this project, and the Virginia Tech library for their cooperation in criticizing MARIAN operation and in providing us with the data on which the system runs.

Bibliography

Borgman, C. L. (1986). Why are online catalogs hard to use? Lessons learned from information retrieval studies. *Journal of the American Society for Information Science,* 37, 387-400.

Chen, Q. F. (1992). An object-oriented database system for efficient information retrieval applications. Virginia Tech Dept. of Computer Science Ph.D. Dissertation, Blacksburg, VA.

Daoud, A. M. (1993). Efficient Data Structures for Information Storage and Retrieval. Virginia Tech Dept. of Computer Science Ph.D. Dissertation, Blacksburg, VA.

Fox, E. A. (1983). Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types. Cornell Univ. Dept. of Computer Science Ph.D. Dissertation, Ithaca, NY.

Fox, E. A (1987). Development of the CODER System: A Test-bed for Artificial Intelligence Methods in Information Retrieval. *Information Processing & Management*, 23, 341-366.

Fox, E. A., & Wilson, L. (1991). Comparison of Advanced Retrieval Approaches for Online Catalog Access. Final report for OCLC grant, University Library, Virginia Tech, Blacksburg, VA. Available as Report ED 338 262 from ERIC Clearinghouse on Information Resources.

Fox, E. A., Chen, Q. F., & France, R. K. (1991). Integrating Search and Retrieval with

Hypertext. In Berk, E. & Devlin, J., Eds. *Hypertext/Hypermedia Handbook.* New York: McGraw-Hill, Inc., pp. 329-355.

Fox, E. A., Chen, Q. F., Daoud, A., & Heath, L. S. (1991). Order-preserving minimal perfect hash functions and information retrieval. *ACM Transactions on Information Systems,* 9, 281-308.

Fox, E. A., Chen, Q. F., & Heath, L. S. (1992). A faster algorithm for constructing minimal perfect hash functions. In *Proc. SIGIR 92, 15th Int'l Conference on R&D in Information Retrieval,* Copenhagen, Denmark.

Fox, E. A., Heath, L. S., Chen, Q. F., & Daoud, A. (1992). Practical Minimal Perfect Hash Functions for Large Databases. *Communications of the ACM,* 35, 105-121.

Fox, E. A., France, R. K., Sahle, E., Daoud, A. & Cline, B. E. (1993). Development of a Modern OPAC: From REVTOLC to MARIAN. In *Proc. SIGIR '93, 16th Int'l Conference on R&D in Information Retrieval,* Pittsburgh, PA, 248-259.

France, R. K. (1991). Lexical Resources as Reference Works and as NavigationTools in an Integrated Information System. In *Proc. ASIS '91, 54th Annual Meeting of ASIS,* Washington, D.C., p. 328

Harman, D. (1992). Ranking algorithms. In W. Frakes & R. Baeza-Yates, Eds., Information retrieval: Data structures & algorithms. Englewood Cliffs, NJ: Prentice-Hall, p. 380-387.

Hildreth, C. R. (1985). Online Public Access Catalogs. In M. Williams, Ed., *Annual Review of Information Science and Technology,* 20, 233-286.

Hildreth, C. R. (1987). Beyond Boolean: Designing the Next Generation of Online Catalogs. *Library Trends,* 35, 647-667.

Kapur, J. N. (1992). *Entropy optimization principles with applications.* Boston: Academic Press.

Kohonen, T. (1977). *Associative memory : a system-theoretical approach.* Berlin; New York: Springer-Verlag.

Kostennieme, K. (1984). "A general computational model for word-form recognition and production." In *Proceedings of Coling '84.* Association for Computational Linguistics, pp. 178-181.

Krovetz, Robert (1993). "Viewing morphology as an inference process." In *Proceedings SIGIR '93, 16th Int'l Conference on R&D in Information Retrieval,* Pittsburgh, PA, 191-202.

Library of Congress. (1988a). Network Development and MARC Standards Office. *USMARC format for bibliographic data.* Washington, DC: Cataloging Distribution Service, Library of Congress, base text 1988; continually updated.

Library of Congress. (1988b). Subject Cataloging Division. *Library of Congress Subject Headings.* Washington, DC: Library of Congress, annual 1988-.

Miller, G. A. (1957). "Some effects of intermittent silence." *American Journal of Psychology,* 70, 311-313.

O'Connor, J. (1980). Answer-Passage Retrieval by Text Searching. *Journal of the American Society for Information Science,* 31, 227-239.

Salton, G. and McGill, M. J. (1983). *Introduction to modern information retrieval.* New York: McGraw-Hill.

Van Rijsbergen, C. J. (1979). *Information retrieval (2d ed.).* London: Butterworths.

Yee, M. (1991). System Design and Cataloging Meet the User: User Interfaces to Online Public Access Catalogs. *Journal of the American Society for Information Science,*

42, 78-98.
Zipf, G. K. (1949). *Human behavior and the principle of least effort; an introduction to human ecology.* Cambridge, MA: Addison-Wesley Press.