Effective, Efficient Retrieval in a Network of Digital Information Objects

Robert Karl France

Dissertation submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science

Edward A. Fox, Chair
Lenwood Heath Eileen Hitchingham
Dennis Kafura Naren Ramakrishnan

November, 2001
Blacksburg, Virginia

Keywords:
class manager, digital library, indexing, information network, information retrieval, MARIAN, NDLTD, searching, stopping rules, weighted sets

Copyright 2001, Robert K. France

Effective, Efficient Retrieval in a Network of Digital Information Objects

Robert Karl France

(ABSTRACT)

Although different authors mean different thing by the term "digital libraries," one common thread is that they include or are built around collections of digital objects. Digital libraries also provide services to large communities, one of which is almost always search. Digital library collections, however, have several characteristic features that make search difficult. They are typically very large. They typically involve many different kinds of objects, including but not limited to books, e-published documents, images, and hypertexts, and often including items as esoteric as subtitled videos, simulations, and entire scientific databases. Even within a category, these objects may have widely different formats and internal structure. Furthermore, they are typically in complex relationships with each other and with such non-library objects as persons, institutions, and events.

Relationships are a common feature of traditional libraries in the form of "See / See also" pointers, hierarchical relationships among categories, and relations between bibliographic and non-bibliographic objects such as having an author or being on a subject. Binary relations (typically in the form of directed links) are a common representational tool in computer science for structures from trees and graphs to semantic networks. And in recent years the World-Wide Web has made the construct of linked information objects commonplace for millions. Despite this, relationships have rarely been given "first-class" treatment in digital library collections or software.

MARIAN is a digital library system designed and built to store, search over, and retrieve large numbers of diverse objects in a network of relationships. It is designed to run efficiently over large collections of digital library objects. It addresses the problem of object diversity through a system of classes unified by common abilities including searching and presentation. Divergent internal structure is exposed and interpreted using a simple and powerful graphical representation, and varied format through a unified system of presentation. Most importantly, MARIAN collections are designed to specifically include relations in the form of an extensible collection of different sorts of links.

This thesis presents MARIAN and argues that it is both effective and efficient. MARIAN is effective in that it provides new and useful functionality to digital library end-users, and in that it makes constructing, modifying, and combining collections easy for library builders and maintainers. MARIAN is efficient since it works from an abstract presentation of search over networked collections to define on the one hand common operations required to implement a broad class of search engines, and on the other performance standards for those operations. Although some operations involve a high minimum cost under the most general assumptions, lower costs can be achieved when additional constraints are present. In particular, it is argued that the statistics of digital library collections can be exploited to obtain significant savings. MARIAN is designed to do exactly that, and in evidence from early versions appears to succeed.

In conclusion, MARIAN presents a powerful and flexible platform for retrieval on large, diverse collections of networked information, significantly extending the representation and search capabilities of digital libraries.

Dedication

For my wife, whose loving support sustained me in my work. And for my children, whose presence in my life has been a blessing. To you, Ruth, Jaime and Katie, I dedicate this dissertation, with my love.

Acknowledgements

The efforts described in this dissertation were built on the work of many people. Influential ideas are credited to many, including Aristotle, Daniel Bobrow, Ronald Brachman, Fred Brooks, Gottlob Frege, Joseph Goguen, Leibniz, Hector Levesque, Marvin Minsky, C.S. Pierce, Dana Scott, Mark Stefik, Leon Sterling, Alfred Tarski, Bernard Witt, and Ludwig Wittgenstein.

Support for projects on which I have worked has been provided by Virginia Tech as well as many sponsors: AOL, DEC, IBM, National Library of Medicine, National Science Foundation (CDA-9303152, CDA-9312611, DUE-9752190, DUE-0121679, IIS-9986089, IIS-0080748, IIS-0086227, IRI-8703580, IRI-9116991, IST-8418877), OCLC, and the Virginia Center for Innovative Technology.

Some of those whose contributions are acknowledged for my early work include: Sachit Apte, Qi-Fang Chen, Lyn Duncan, Lee Hite, Muriel Kranowski, Ken Laws, Marie-Lise Lasoen, J. Martin, David Miller, Joshua Mindel, John Roach, Mahasweta Sen, Sharat Sharan, Bruce Smith, Mark Tischler, Joy Weiss, and Bob Wohlwend.

Thanks go to many colleagues, including R. Akscyn, F. Can, J.H. Canós, J. Carroll, V. Chachra, M. Crowder, T. Doszkocs, D. Dudley, J. Eaton, J. French, H. Hartson, L. Heath, E. Hilf, D. Hix, M. Hohlfeld, B. Kim, J. Lee, M.-H. Lee, A. Maeda, G. Marchionini, G. McMillan, J. Nutter, Y. Qian, D. Rao, H.P. Reddy, M. B. Rosson, T. Severiens, C. Snavely, S. Urs, M. Williams, J. Young, and K. Zimmermann.

Thanks also go to many students, including G. Abdulla, A. Atkins, R. Barnhart, S. Betrabet, D. Brueni, B. Cline, A. Daoud, F. Das Neves, S. Datta, R. France, M. Gonçalves, S. Guyer, W. Heagy, N. Kipp, M.P. Koushik, S. Kriss, A. Krowne, W. Lee, B. Liu, N. Liu, P. Mather, J. Menezes, A. Narasimhan, L. Nowell, G. Nunn, A. Pande, S. Patel, C. Phanouriou, J. Powell, R. Quizon, R. Richardson, V. Riggins, E. Sahle, J. Shaw, P. Sheldon, O. Sornil, M. Subhas, H. Suleman, W. Wake, M. Weaver Smith, J. Wang, S. Winett, W. Xi, C. Zhang, and Y. Zhou.

Additional thanks go to those who are acknowledged for particular roles as discussed in sections at the ends of chapters 3 and 4.

Special thanks go to my committee, Edward Fox, Lenwood Heath, Eileen Hitchingham, Dennis Kafura, and Naren Ramakrishnan.

[Addendum by chair of committee:
Special thanks go to Robert's family, friends, and colleagues, who helped make it possible to complete and approve this document while Robert was critically ill in the hospital. I am sorry that it was not possible to provide a conclusion, to unify the page numbering of chapter 4, or to complete other improvements that Robert would have made if he were able. – Edward A. Fox]

Table of Contents

Chapter 1: Networks of Information Objects

We begin with a simple model from graph theory: a *network* is a set of nodes together with a set of (directed) edges, or links. To this model we add two enhancements. First, we insist that both nodes and links participate in class systems of the sort familiar from object-oriented programming languages. The node class system and the link class system do not interpenetrate, and neither one adds complexity to the underlying graph-theoretic model, so the total complexity of the model is the maximum complexity of any of the three structures. All of these structures are compact and computable, and common limitations to the sorts of graph and class operations provided keep the system as a whole tractable. In fact, strategic use of the class system can often greatly simplify graph operations, since it can greatly restrict both the number of nodes and the topology of subgraphs that must be considered.

The node and link class systems carry the semantics that make graphs *information* networks. For instance, we make use of class inheritance within the node system to define and refine what it means for a node to bear information content, thus providing a representational realization of that key ingredient in a digital library, the *digital information object.* Digital library collections are made up of such objects, whether they are pieces of formatted text, streamable video segments, hypermedia constellations, or terabyte scientific databases. Each category of digital information objects can be treated as a class of information-bearing nodes, using the normal techniques of object-oriented programming to develop the personality of the category in terms of presentation and manipulation methods for the information objects in that class. (Some categories of information objects can also be profitably [treated / represented] as subgraphs within the digital library information network, but we will postpone this topic to a later section.)

We will regard a class of nodes as information-bearing if it is describable: if it can be said to match simpler information objects at some measurable level. For instance, a piece of text can be said to match a query string using measures from edit distance to term-vector cosine; a MIDI file can be said to match a set of instrument selections or a pattern of notes; and a metadata record can be said to match a specification of authorship and/or subject. In computational terms, information-bearing nodes are objects that support a method that calculates how well they match input descriptions of the correct type. More complex information objects require more complex descriptions and matching methods, but the method prototype – the abstract match() method of the parent class *InformationObject* – remains the same.

The output of matching methods is the motivating use for our second representational enhancement. We enhance the node-link model by allowing any node or any link to carry a *weight*. Weights are defined axiomatically to correspond to informal concepts of "importance," "uncertainty," or "goodness of fit". Given the axioms, a simple realization for weights is the real interval [0...1].

Weights are not a new concept, but occur in information retrieval systems, evidentiary and approximate reasoning systems, and fuzzy logic. The prime differences among these systems are not the raw objects, but the particular operations supported. With some additional axioms, weights can also be refined into probabilities. This is important when we consider weights derived from statistical and information-theoretic characteristics of the information network.

Properly speaking, weights only have meaning in comparison to other weights. The germinal concept here is the *weighted object set:* a set of objects whose relationship to some external proposition is encoded in their decreasing weight within the set. For instance, we can ask "What English words might the string 'theis' be a corruption of?" The method for evaluating that could use a spell-check routine that applies distance metrics on a collection of known English words to recover such possibilities as 'their,' 'thesis,' 'theistic,' and so forth. The end result would be a weighted object set of English words, where those words at greater distance would receive lower weight.

Weighted sets also arise as an expression of uncertainty, for instance when attempting to identify a name in a citation with one of a set of possible authors in an authority list. Weighted classes within the node and link class systems are meaningful as a direct application of fuzzy set theory [Zimmermann]. We find it useful to allow weights to be determined based on class semantics, rather than enforce a single global model. An example is the class of weighted *OccursIn* links that connect terms to the texts in which they appear, and whose weights are functions of the link class topology.

These two extensions to the basic node-link model are sufficient for our purposes in the design and analysis of the MARIAN search system. We note, however, that they are not necessarily adequate to the digital library domain in general. For instance, a digital library might well choose to associate descriptions of provenance and pedigree with each object and each link in the system just as we do weight. We see no formal reason to avoid such additions and expect to add these enhancements to future implementations of the model.

1.1 Searching the Network; Searching the Class System

A digital library built on the information network model described in the last section could easily comprise hundreds of millions of objects and billions of links. We simplify this system by looking not at the individual links and nodes in a collection, but at the link and node classes. This can reduce the cognitive complexity of digital library operations by many orders of magnitude. Whether the computational complexity of the operations are similarly reduced depends on class semantics. An example may help to make this clear.

Consider a very simple digital library collection consisting solely of *Works* and *Persons,* where *Persons* are related to *Works* only as authors. If the collection followed the overall shape of the Virginia Tech Library collection it would contain millions of *Works,* hundreds of thousands of *Persons,* and multiple millions of *HasAuthor* links (Fig. 1.1). We refer to this sort of representation as an *instance-level* network.

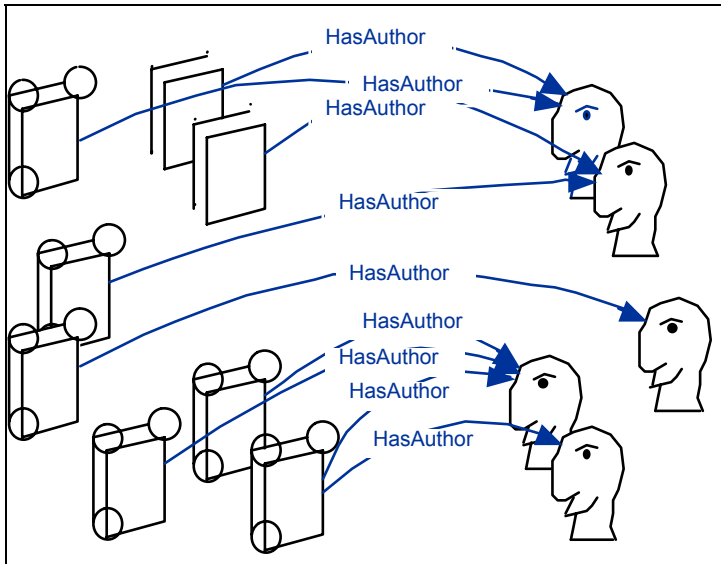


Fig. 1.1: A small collection of *Works, Persons,* and *HasAuthor* links. (Note distribution of *Works* among *Persons,* including one work with two authors at bottom.) Even this small collection appears complex with this representation.

Suppose that we want to search this network by mapping an author's name to a (weighted object) set of related works. One way to perform the search, advocated by certain schools of object-oriented programming and highly parallel distributed information processing as well as by proponents of the SODA model [Nelson] is to broadcast the name requested to all *Person* nodes. Any node that recognizes the name as its own (or perhaps a close match to its own) would then "activate" all *HasAuthor* links connecting it to *Works,* which will in turn result in those *Works* presenting themselves to the originator of the search.

This model has many conceptual advantages, particularly for highly parallel architectures. However, it does involve operations by all the hundreds of thousands of *Person* nodes in the collection, as well as any *HasAuthor* links and *Work* nodes activated. One must also consider the cost of the initial broadcast as well as the multiple presentation operations, each of which involves some fixed transmission cost, and the cost of assembling and ordering the *Works* presented. Finally, there is an administrative, human cost to designing, debugging, and monitoring a system of this complexity.

There are also human interaction issues to be considered. In a large-scale digital library, simple query descriptions like "**Author:** J. Smith" can match extremely large numbers of works. In an instance-level network, the number of works resulting from such a query can be controlled by limiting the activation level required for a node to present itself. However, appropriate limit values vary from user to user and from task to task. In fact it is not unusual for a user, having seen the most active nodes, to choose to extend her search to less active nodes. Provisions can be made to skim the high-activation nodes from a full set of activated nodes, but it is hard to avoid costs of activating unused nodes on the one hand, and of collecting further "strata" of lower-activation nodes on the other.

In contrast, MARIAN considers digital library collections – and all information networks – at a higher level of abstraction, one step removed from the preceding model. Our collection of

authored works in MARIAN becomes a graph with two nodes and a single link, where each of these objects functions as manager to an entire class of instances (Fig. 1.2).

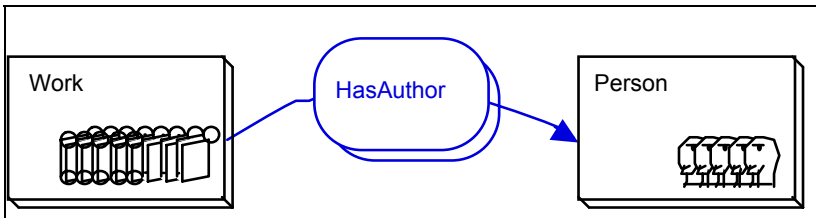


Fig. 1.2: Organization is clearer and search efficient when the collection is viewed as a structure of classes.

Each class in a particular library collection is the responsibility of a class manager, elsewhere called a class factory. Class managers store and maintain instance objects of their class, translate back and forth between object IDs and fully realized objects, and participate in the class hierarchy by collecting results from their subclasses and responding to requests from their superclasses. They also function as the search engines for the system.

The advantages of this model are more than cosmetic. First, there is no longer any need to broadcast a query description throughout the network. In our example query, rather than "J. Smith" being broadcast to all *Person* class nodes, the description is directly passed to the *Person* class manager. Next, the cost of finding *Persons* matching the description is now determined by the interior structure of the class manager rather than the overt structure of the information network. Since the class manager is privy to the common structure of its instance nodes, it can more optimally implement search and matching functions. For simple objects like named persons, techniques such as hashing and B-trees suggest themselves; in more general applications, class managers can make use of sophisticated database techniques to optimize the search process. Any such algorithmic savings is a win over processing by distributed instance nodes.

Link and node activation still occurs in a class-level network. Activation at the class level consists of passing weighted object sets of activated nodes among the class managers, where set weights correspond to node activation levels. In itself, this is neither a gain nor a loss over instance-level activation. It becomes a gain in MARIAN only through lazy evaluation of sets, discussed below. On the other hand, class managers make possible a considerable semantic simplification. Semantics of link activation – for instance, what happens when two authors of a single work both match the query description – can be efficiently implemented by the link class manager, rather than being handled by the common target node. Since different sorts of links have different combining semantics (see next section) this removes a burden of complexity from node design and centralizes it squarely where it belongs: in the implementation of the appropriate link class.

Presentation of results is also simplified. In an instance-level model, activated objects present themselves to the user. There are at least three problems with this rubric. First, in any realistic digital library, information objects are likely to have several possible presentations. Thus before each activated object can present itself, it must enter into a dialog with the user (or more likely,

some surrogate of the user) to establish which presentation to use. Second, as mentioned above, users are unlikely to want to examine every object activated – at least at first. Third, activated objects need to be presented to users in some reasonable order. Activation level (goodness of fit to the query description) is certainly one option, and is a natural in the instance-level model. When activated objects report individually to the user (surrogate), however, further processing is required to put the full set into order. Moreover, goodness of fit is not always the best presentation order in a digital library. Depending on the context, users may be better served by bibliographic order or recency of publication, among others. All of these difficulties can be addressed when presentation is moved to the class manager level. With its global perspective on the structure of instance objects and on the full weighted set of activated objects, the class manager is in the best possible position to make decisions on how many objects to present and how.

Finally, the MARIAN class-level model offers opportunities for lazy evaluation not available in the instance-level network. Recall the query "**Author:** J. Smith." A user authoring this query may be satisfied by examining a few nodes, or she may need to explore the result set to an arbitrarily low level of match before her task is accomplished. In an instance-level system, activation either occurs or does not occur, depending on the activation level set. There is no easy way in such a system to defer activation until needed. In contrast, a class-level system need only develop as many candidate results as the user requests. In MARIAN this is accomplished by maintaining weighted object sets as abstract objects that are evaluated only as needed.

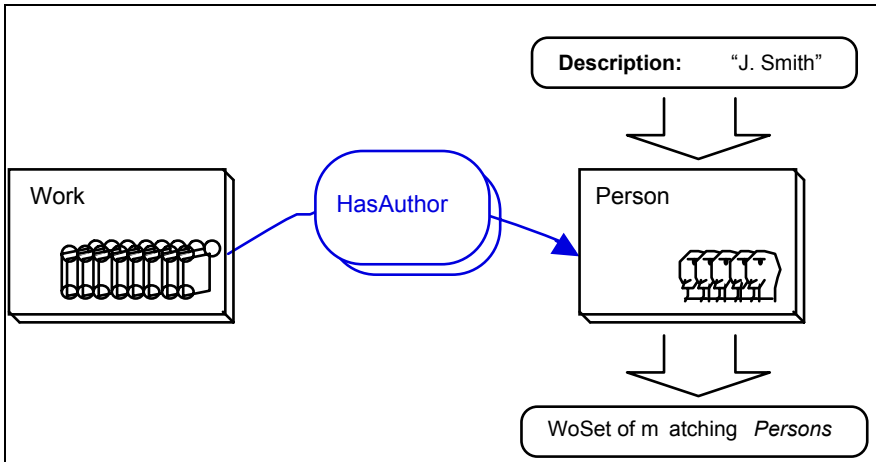


Fig. 1.3: The *Person* class manager maps a description to a weighted object set of matching *Person* objects.

Consider a likely scenario for our authored-work collection. Image that the *Person* class manager uses database operations to identify persons whose names match descriptions like "J. Smith." Performing these operations results in a large and fully-instantiated weighted object set of *Person* nodes (Fig. 1.3). (Other techniques might not require even this initial set to be fully instantiated, but we will start this example this way because fully instantiated database result sets are a common starting point in MARIAN systems.) The *Person* class manager does not, however, pass this full collection of instances to the *HasAuthor* manager, but instead sends an abstract weighted object set as a surrogate. The *HasAuthor* manager in turn creates an abstract weighted object set of *Work* nodes to pass to the *Work* manager. This new set is based on the set

of *Person* nodes, with the property that every *Work* in the new set is linked to at least one *Person* in the incoming set, but is passed in a completely unevaluated state (Fig. 1.4).

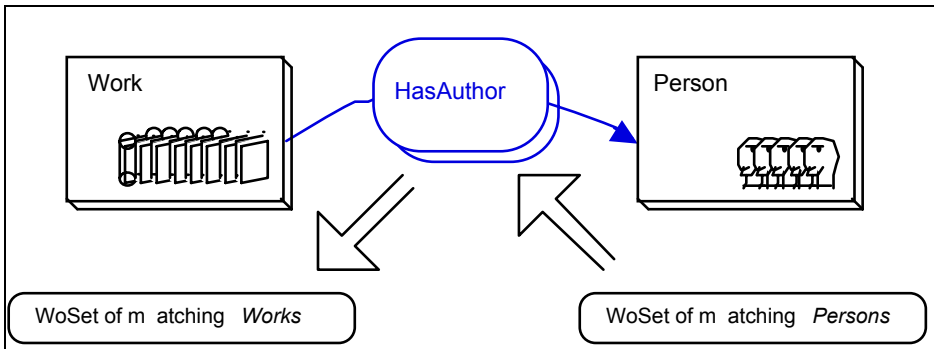


Fig. 1.4: The *HasAuthor* link class manager generates a surrogate for the corresponding weighted set of *Work* objects, but does not actually generate the objects.

The *Work* manager is now in a position to prepare a result set presentation for the user (Fig. 1.5). Often the manager will know in advance how many matching works to show the user and in what form and order; other times it may work from personal or system defaults. Let us say that it is to prepare the twenty best matches. Accordingly, it asks the incoming set surrogate for its top twenty members. The surrogate uses *HasAuthor* class methods and data to develop these from the elements of the original weighted set of *Persons* (Fig. 1.6). This may involve requesting as few as one *Person* if the top ranked person authored as many as twenty works, as many as twenty *Persons,* or even more if many top-ranked people are all joint authors of the same works. The number of instances requested, and thus the transmission costs across the network and the amount of work done by the *Work* and *HasAuthor* managers, are severely limited compared to the size of the original *Person* set.

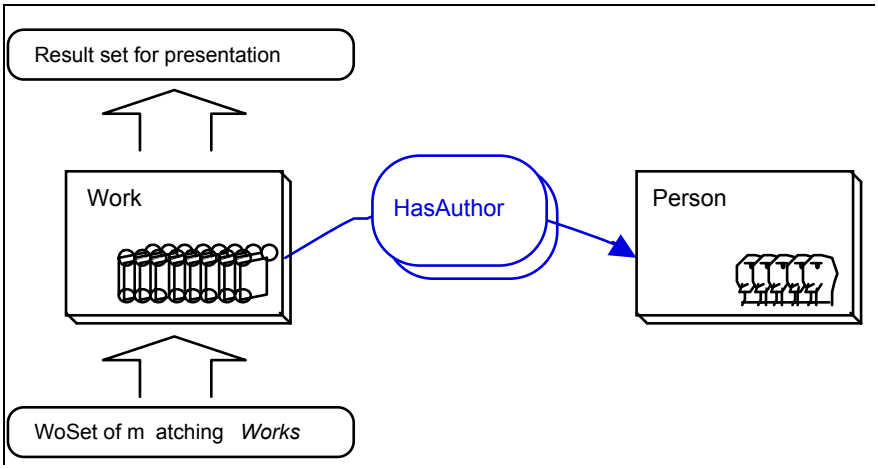


Fig. 1.5: The *Work* class manager can now generate a presentation for the user. In a more complex collection, the *Work* manager might also combine the incoming set with other descriptions such as title text or subject.

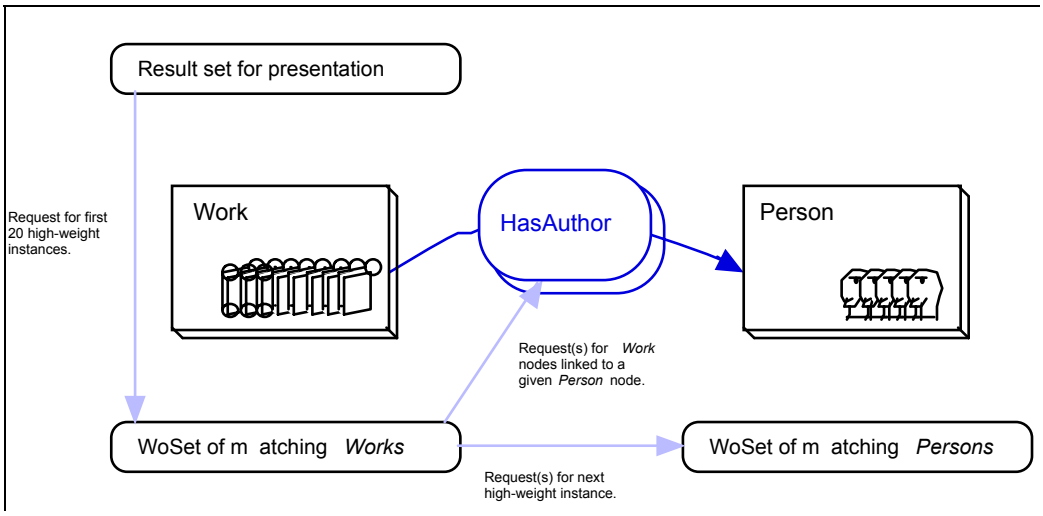


Fig. 1.6: Requests on the *Work* set cause incremental evaluation of set contents mediated by corresponding requests on the *Person* set and on the *HasAuthor* class manager.

In this way the class-level system exerts minimal energy to produce a presentation set for the user. Should the user choose to expand the set, the structures are already in place to continue the evaluation. This sort of lazy evaluation is extremely important in a digital library environment, where different users have extremely different needs, from known-item search for a single work to in-depth research exploration, and where these needs cannot be distinguished based solely on overt user behavior.

1.2 Why Use Links?

On the other side of the MARIAN class-level system, in a sense, are information-retrieval systems that consider objects alone without links. ... Another venerable example of this form of representation is the MARC record. ... While MARC records have rarely been used in this format for functional library systems, the very representation demonstrates some of the problems with this approach.

Separating digital library objects into classes is an obvious improvement on this approach that has lead to such systems as RDF [Lassila] and the Dublin Core initiative [DCC]. As mentioned above, regarding *Persons* and *Works* as different classes of object allows search methods – and the underlying formula for similarity of a class instance to a query description – to be tailored to the structure and semantics of the objects in the class. Presentation and storage methods can also be specialized in a class hierarchy – and are, in many modern library catalog systems [Virtua, LoC, ...]. It is nonetheless often the case that they fail to store relationships between objects explicitly, and instead store references to related objects within class instances. MARC author fields (1xx, 7xx) can be replaced with references to authority file records, and associates with each authority file record a [list] of references to authored works.

Representing objects explicitly but relationships implicitly violates an essential principle of knowledge representation: store knowledge in declarations, not procedures; as structures, not operations. Only knowledge that is represented explicitly is available to certain kinds of operations. It is very hard to examine the topology of the authorship relation – what patterns exist among authors and works – while the information is implicitly encoded. Even determining average works per author and average authors per work is non-trivial. And it is impossible to point to an authorship relationship: to make statements like "We believe that Marlowe is the author of *The Damnation of Faust* because ..." or "The fact that one can learn that Walter Scott and 'The author of *Waverly'* describe the same person is an important example in the theory of identity."

There are also pragmatic reasons to make relationships explicit in a digital library. First, data that is held explicitly is much easier to police. When the authorship relation is explicit there is no possibility that a *Work* may appear in the reference list of a *Person* whose name does not appear in the MARC record of the *Work,* or of the sort of "dangling hyperlinks" that so bedevil the Web. Consistency checks can be initiated on the level of authorship, as well as on the level of works and persons. Second, explicit links can be semantically coded to behave in different ways depending on their class. Third, explicit links are more flexible than implicit links. They can, for instance, be run in reverse. Thus it is possible in MARIAN to move from a work to its author, from a person to all authored works, or from a work to all other works with the same author. It is even possible to move from a relevant work or set of works to a complex graph describing the common characteristics of those works, and thus to other works that match that graph. Finally, explicit links are amenable to structural change as the digital library grows and changes.

1.3 Growth and Change in Digital Library Networks

One of salient characteristics of digital libraries is that they are constantly being modified. This is a special case of Ranganathan's fifth law of library science: "A library is a growing organism." Libraries and especially digital libraries grow, and not only in size. They also grow morphologically, as new demands on their function create newer and more complex structures within the old. We can refer to this development as "materializing structure". Materializing structure is a very powerful technique, as an examination of part of MARIAN's history will demonstrate.

MARIAN's first use was as an online library catalog, supporting searches on MARC metadata records. In representing this domain, we set up three initial classes: *Works,* realized here as MARC records; *Subject Descriptors,* and *Individuals.* This last class included all the sorts of things that librarians accept as authors: natural persons, corporations and similar organizational entities, and scholarly conferences. *HasSubject* and *HasAuthor* links connected the node classes (Fig. 1.7). These five classes completed the library information network in its first realization, but were not the only classes in the system network. The complete network also included information system classes designed to facilitate searching on text. Each category of library object had at least one text field. Each type of individual had a unique *name;* subject headings

had a text *descriptor;* and we described *Works* as having two text components: *title* and *notes*. All of these text fields were synthesized from more complex MARC fields – in the case of *title* and *notes*, from several fields concatenated together – but each was regarded as a single piece of text for information retrieval purposes.

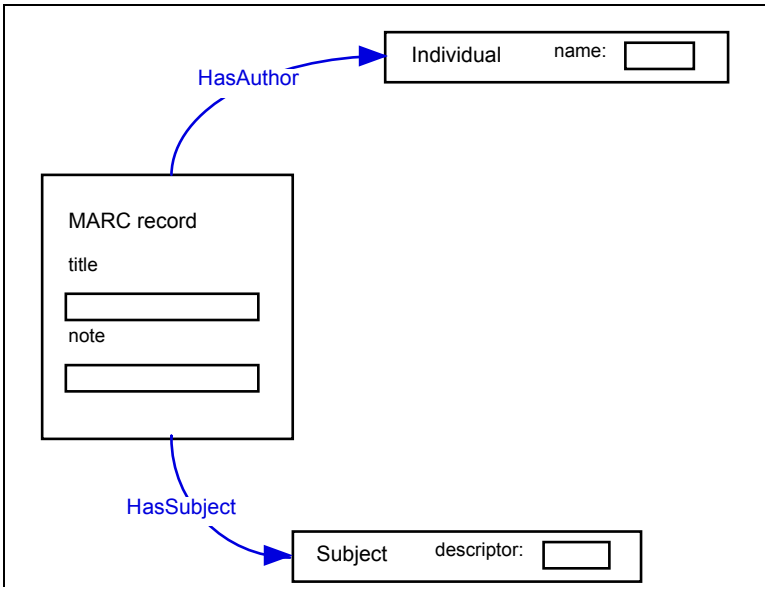


Fig. 1.7: The initial MARIAN ontology was composed of MARC metadata records (representing library works), individuals and subject headings, linked by HasAuthor and HasSubject relationships.

Text is a particular sort of information object, and like any class of information object, supports similarity and search methods. In the case of text, those methods depend on substructures – usually words or other terms – within the text string and the query description. This early version of MARIAN used a fairly typical approach to text matching: to compute the similarity of a text to a query, first break the text and query into bags of terms and determine which terms occur in both bags, then compute similarity based on how much of the query falls into the overlap, how much of the text falls outside of the overlap (the amount of *noise* in the text), and how important the terms are in the total collection, statistically speaking.

Text matching and search operations thus reduce to operations on terms and on the *OccursIn* relationship, represented in MARIAN as a class of weighted links (Fig. 1.8). A text query is searched by "chasing links" from the query description, through *OccursInQuery* links to the *Term* class, and thence through previously generated *OccursIn* links to the appropriate class of *Text* object. In the process, final query-text similarity weights are generated from the link weights and the connection statistics for the nodes encountered, most of which were pre-computed at the time of the last collection update (Fig. 1.9).

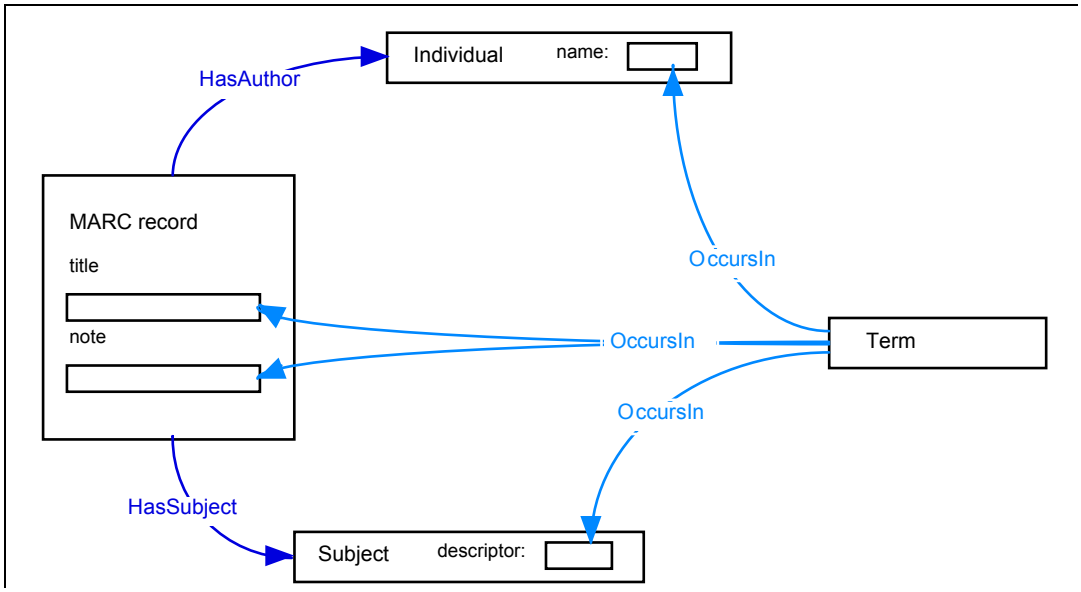


Fig. 1.8: Each text field is linked to the terms that occur within it – or more properly, the terms are linked to the texts.

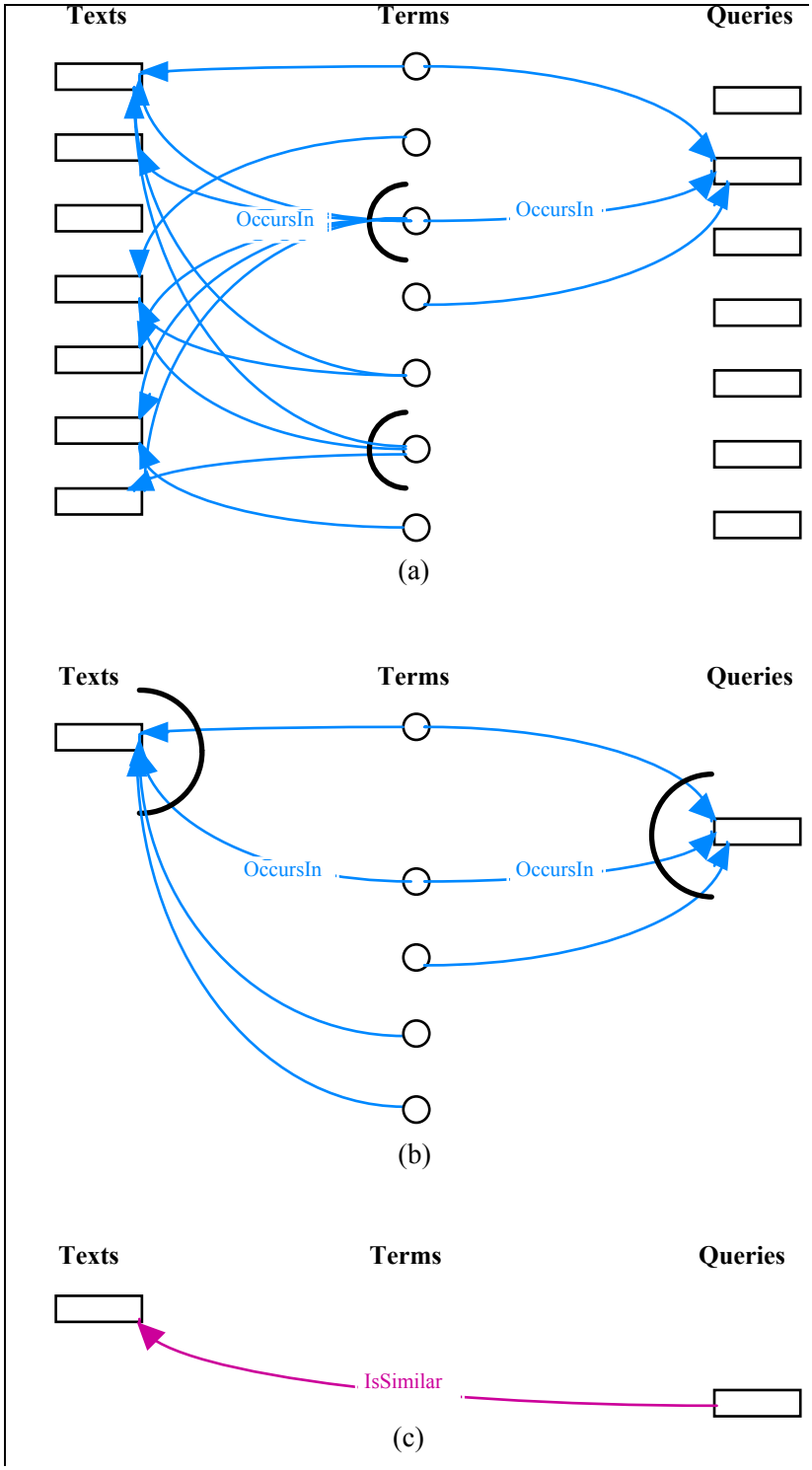


Fig. 1.9: Instance-level view of the links between texts, terms, and queries. Part (a) shows a portion of the collection-wide context. Bold arcs here show where the importance of a term in the collection is calculated based on its outdegree. In (b) we consider only a single text-query pair. Bold arcs describe the indegrees of text and query: the fraction of each included in connecting paths is used in calculating text-query similarity. The result is a view where a single weighted *IsSimilar* link connects the text and query (c).

The class *Term* is of a different sort than any so far considered: it is an arbitrary union of disjoint subclasses. *Term* functions as a superclass to classes that have little to do with one another except that they can all be discovered as components of text. Specifically, in most versions of MARIAN *Term* is the combination of *EnglishRoot, EnglishInflectedForm, Integer, DecimalNumber, CodeNumber* (strings like "077-46-8013" or "235a/IEEE903.2") and the catch-all class of *CharacterString.* This collection of classes was inherited from CODER [Fox&France], which also added *ChemicalFormula* and *Fraction* into the mix. Figure 1.10 shows the proportion of each subclass in Virginia Tech Library titles.

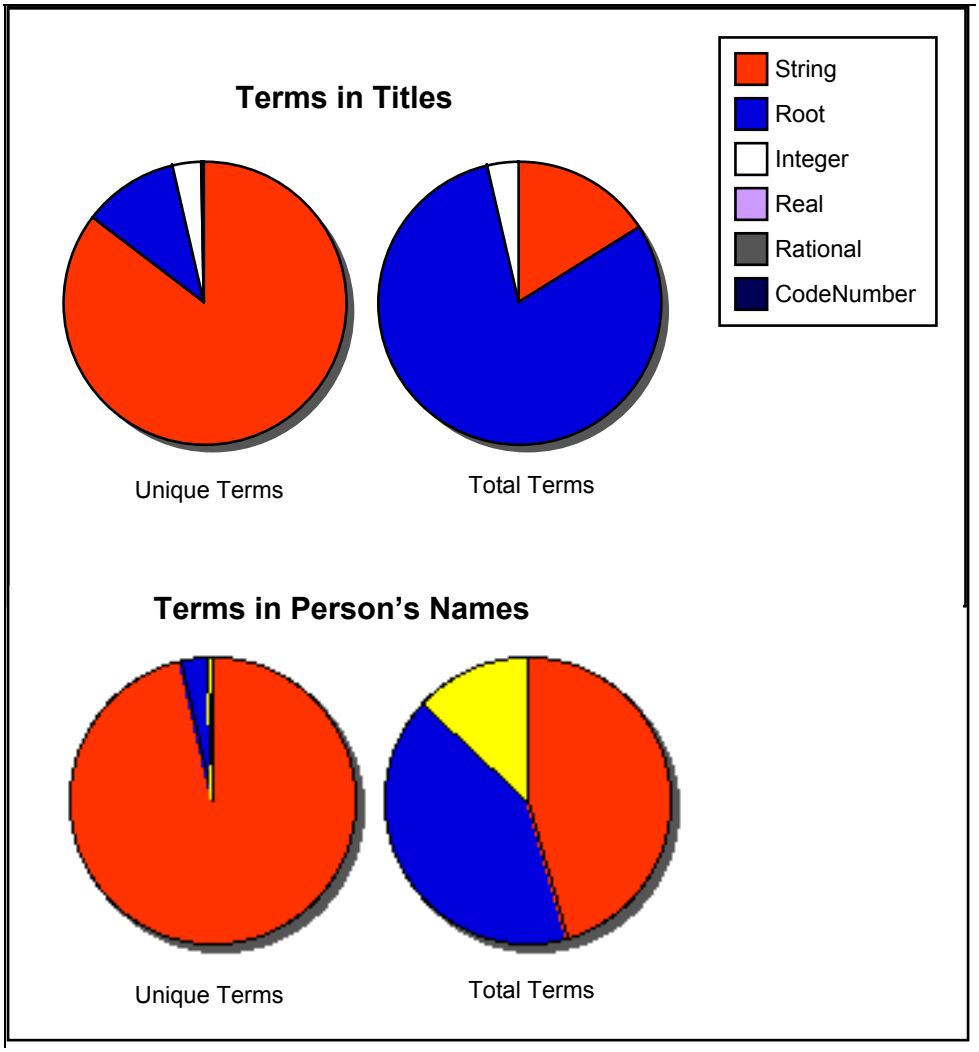


Fig. 1.10: Distribution of terms within VT Library MARC title and person's name fields. Charts on the left show distribution of unique terms; on the right, of all term tokens.

So *Term* is a superclass, but it is not a "natural" superclass. It is instead the union of several classes picked arbitrarily to match our information system design. Such unions are guaranteed by the semantics of the type lattice. They are particularly clear when the subclasses are disjoint, because then each member of the union can be assigned unambiguously to a single subclass. Because it is arbitrary, *Term* is extensible. If collections require additional classes – say, XML tags or words in additional languages – they can be easily added to the union. The main use for the class *Term* is to provide a common terminator for *OccursIn* links (Fig. 1.11).

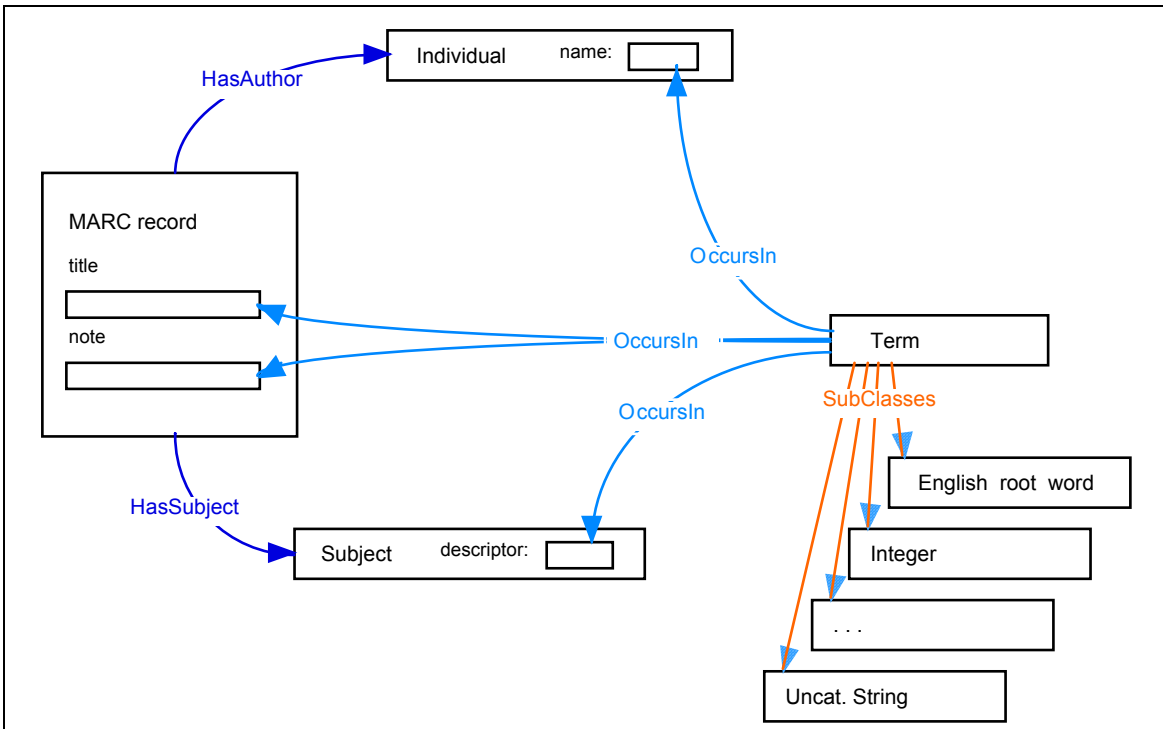


Fig. 1.11: Although *Term* is only a disjoint union superclass with no elements of its own, it still functions as the common source for all *OccursIn* link classes.

Note: In this and subsequent diagrams we indicate class relationships with lines that are straight, dashed, and when viewed in color, orange. Class relationships are not part of the explicit link network (see Chap. 2). Network links will continue to be represented by solid, curved lines in other colors.

The early MARIAN system in this example maintained several distinct classes of *OccursIn* links, one for each text field among the library object classes. All the link classes could be regarded as "natural" subclasses of a parent *OccursIn* link class, but they were stored as drawn in Figure 1.11 by design.

Clearly, the design in Figure 1.11 is not the only one possible. Figure 1.12 shows several other ways that the representation could have been designed. All these structures are semantically equivalent – they encode the same information and the same conclusions can be drawn from all of them – but they differ operationally. In particular, they involve tradeoffs in efficiently matching individual text fields versus matching a description to *any* text field, and in efficiently matching *all* terms versus efficiently matching only a single sort of term. In the case of MARIAN's library catalog application, we wanted to maximize the efficiency of matching all sorts of terms within individual text field classes like *title* or *name.* We expected that this would be the most common search operation requested. So we chose the design in Figure 1.11 to enable a single link class operation to connect the *Term* union class with one field at a time. Searching in several text fields at once was implemented as a structural union operation, described in Chapter 2.

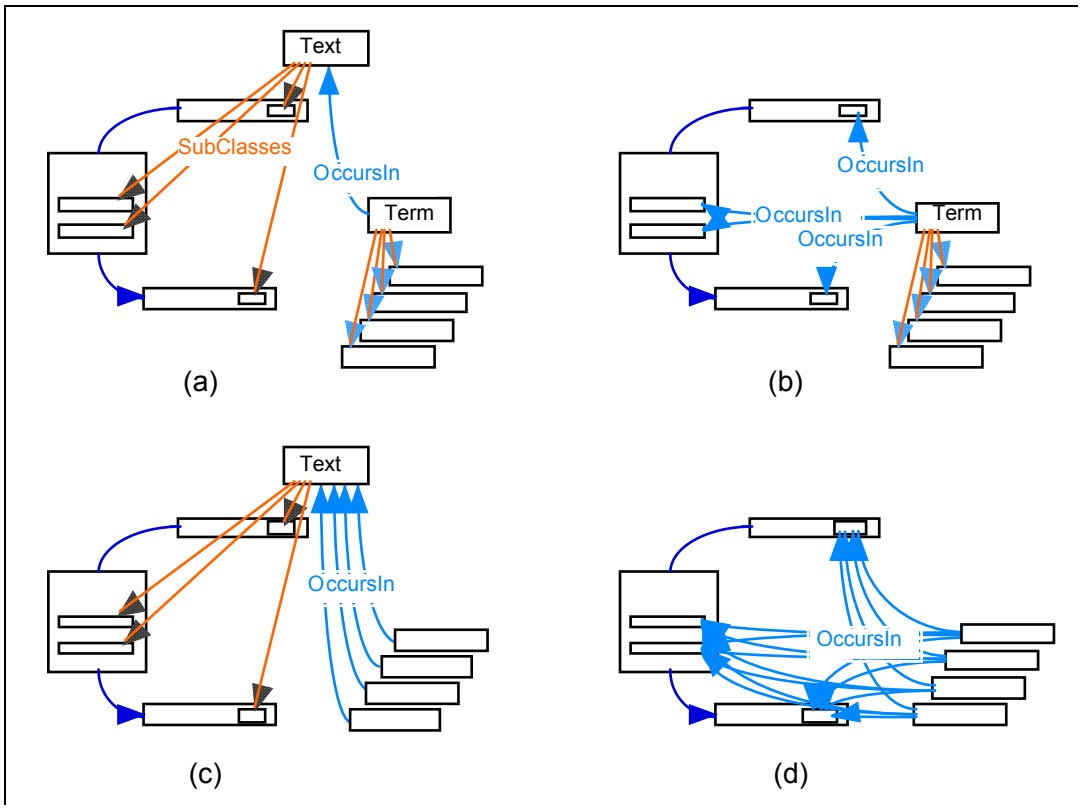


Fig. 1.12: Alternative class structures for the information in Figure 1.11. In (a), both texts and terms are collected into disjoint unions and only a single class of *OccursIn* links is created. (b) shows the chosen alternative, and (c) the dual: a single *Text* superclass is linked to fully analyzed *Term* subclasses. Finally, (d) shows a design alternative with no superclasses at all.

MARIAN v. 0.7 - 1.2 used the set of node and link classes in Figure 1.11. In Chapter 6 we examine the system in more detail. In the meantime, we continue to follow the system's evolution. The categories of Figure 1.11 were sufficient to provide the functionality we were seeking in the OPAC application. As we gained familiarity with system operation, however, we inevitably discovered inadequacies.

One notable feature of MARIAN implementation from the beginning has been a generalized logging facility. Along with obvious uses in system tuning and debugging, the log files have enabled us to track and compare queries.[1] Comparing users' actual behavior to the expected behavior for which the system was designed has been invaluable in evolving MARIAN's functional capabilities. It has also contributed to the morphological evolution of the collection.

An early observation from query logs was that a large proportion of **Author** queries were in fact intended to be limited to personal author, rather than to any form of individual. Searching these against a composite *Individual* class had several unfortunate consequences. First, the search operations were relatively inefficient, because the sets being created and combined were large

[1] Out of concern for users' privacy, these query logs have always been carefully disassociated from information that could be used to identify particular queries with individual users or originating machines

compared to sets of strictly personal authors. On a more subtle level, the statistics used to calculate goodness of fit suffered. Terms in *Individual* names were drawn from several distinct statistical distributions. In particular, conference and corporate names are drawn from distributions quite close to the generalized distribution for English text, but personal names reveal quite a different distribution (compare Figure 1.10). This had a perceptible effect on the order in which MARIAN presented results to the user, causing some authors to appear unexpected high or low in the ranking.

Most importantly, however, combining the different kinds of *Individuals* into a single class made trouble for the users. Two actual examples illustrate the problem. First, consider the user who searched for works by Sam Houston against a class that included government documents produced by the city of Houston, Texas. While MARIAN correctly promoted the governor and hero to the front of the user's retrieval set, the ensuing flood of items – many of which seemed to be on almost the correct subject – created a serious distraction and potential confusion.

Searching for "**Author:** Jane Austen" produced an even worse situation. The problem here was the existence of the Jane Austen Society among corporate authors and a Jane Austen Bicentennial Conference among conference authors. Due to vagaries of representation in the library catalog records and the way MARIAN extracted and assembled names from record fields and subfields, the Jane Austen Society was calculated as having *less* noise than Ms. Austen's name, so works by that "individuals" were promoted above those by the (presumably intended) person. Fortunately, the VT library collection at the time contained only a few Jane Austen Society and Jane Austen Conference works, so the body of works by Austen herself was clearly visible in presentations, but the situation was still clearly unacceptable.

The solution was to break the class of *Individuals* into classes of *Persons, Corporations* and *Conferences,* generating distinct subclasses of *OccursIn* and *HasAuthor* links for each subclass (Fig. 1.13). The *Individual* and *HasAuthor* superclasses were maintained in the representation as disjoint unions of their respective subclasses. Keeping the superclasses meant that users could still perform combined author queries. The cost for combined queries in the new system was slightly higher than in the undivided system, but the statistical calculations were now of higher quality and the result set presentation better. In addition, the superclasses could use vernier weights to balance the subclasses for precise tuning of system performance. Most important, personal, corporate, and conference author queries could now be offered to the user with an increase in effectiveness and a decrease in cost.

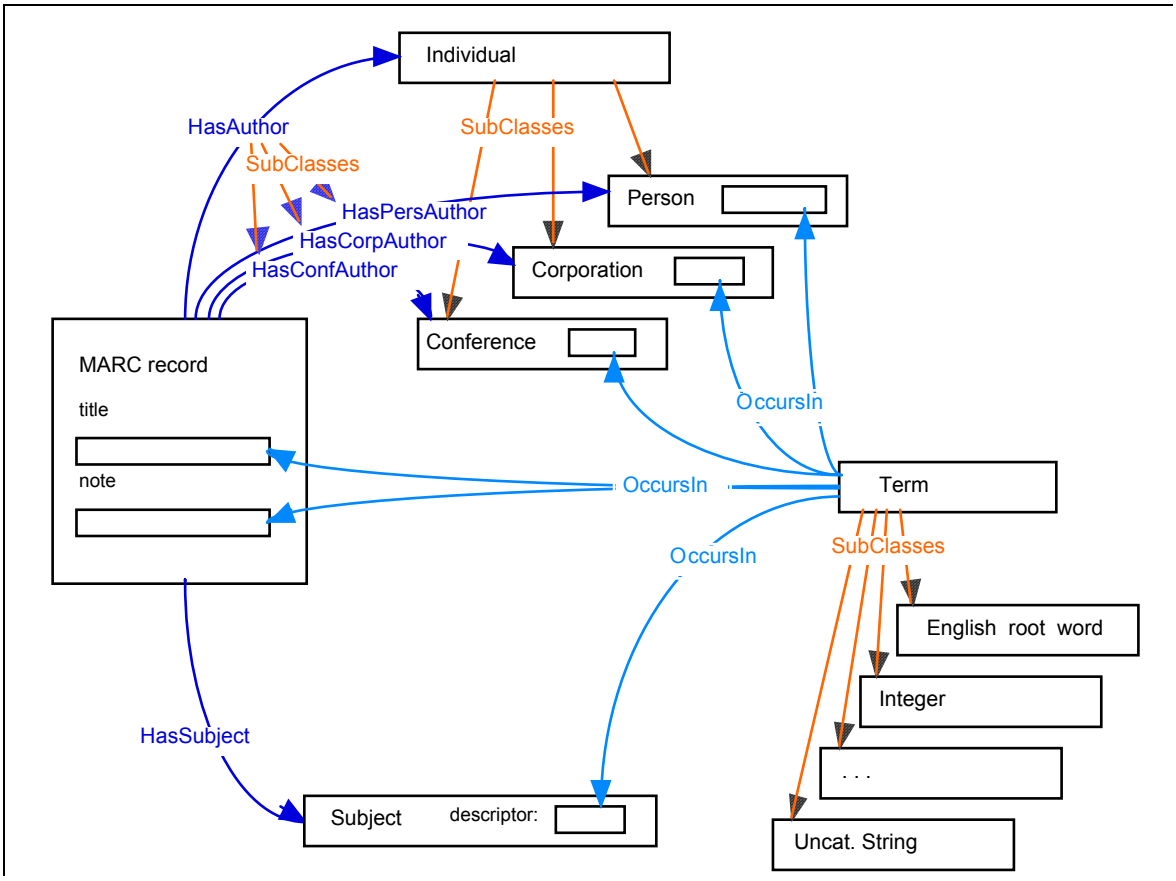


Fig. 1.13: Splitting the *Individual* class into disjoint subclasses produced a more complex network but faster and more effective performance.

The *Subject* class was also split a few years later, but in a different way and for different reasons. In this case, we were addressing a conflict in subject representation and a related problem in subject search effectiveness. To explain this conflict we need to divert a moment and consider library catalog records and cataloging practice.

As mentioned above, the standard MARC catalog record is composed of fields and subfields. Several of these fields hold subject descriptors, and all subject fields can be repeated as necessary to describe a work that covers multiple subjects. As an example, Christiansen's veterinary work *Reproduction in the dog and cat* receives, among others, the headings "Cats -- reproduction" and "Dogs -- reproduction." The headings are entered as two separate MARC fields, since Christiansen is describing only dog and cat surgery, and not "Veterinary medicine -- reproduction" or even "Reproduction -- laboratory animals" (the closest points in the LCSH cataloging hierarchy that I can find that subsume both dogs and cats). Professional catalogers spend their workdays making tricky calls on how to describe works like this within the fixed taxonomy of the LCSH and develop a fine sense of how to use it in describing works.

Unfortunately, users are often completely unsophisticated in cataloging practices and subject taxonomies, so a user looking for veterinary books covering cats and dogs is likely to create the query "**Subject:** cat dog," rather than "**Subject:** cat + **Subject:** dog." With this user in mind,

we originally created the library collection network by concatenating all subject fields for each work into a single field and generating *OccursIn* links for that field.

This approach works well for Dr. Christiansen and the user searching for cats and dogs. It fails miserably for the user searching for "**Subject:** Baptists -- United States -- History" (this is again a real user, in this case one who *was* sophisticated in cataloging practices and who submitted an email comment on this very problem). The problem was not that MARIAN could not locate works with the words "Baptists," "United," "States," and "History" in the subject descriptor text. Rather it located too many works with those words: not only those where the words occurred within the same MARC subject field, but also works where the words occurred in separate and unrelated subject fields: "Baptists -- biographies" and "Biography -- United States -- History," for example.

Two additional effects exacerbated the problem further. Once again, the issue of noise arose. Works with more and more diverse subject entries were calculated as having greater noise than works with fewer and more redundant entries, and so were demoted in the result set presented. And while it can be argued that in general this is reasonable behavior – more and more diverse subject headings should in general indicate a work less specifically focussed on the subject being searched – in the case of a user searching for a specific and correct subject heading the behavior is merely confusing. Second, the VT library abounds in book with "United States -- History" somewhere in their subject. The sheer volume and variety of these, together with the confounding issue of noise, combined to create a situation where some works on simply "United States -- History" were presented before works on "Baptists -- Tennessee -- History." The user's comment that works on "Baptists -- Tennessee -- History" were actually more relevant was particularly stinging, since MARIAN's similarity functions would have agreed ... if they had been run on individual subject entries rather than on the composite text.

We were faced with a dilemma. Treating each subject entry as a separate text would help one category of user, but leave the other in the cold. Combining all subject entries into a single text helped many users, but made life impossible for others. Yet the two seemed mutually exclusive. What to do?

The breakthrough came when we abandoned the idea of using a single representation and decided to create redundant representations for the same subject data: to generate both individual subject entry fields and a composite subject field for each record (Fig. 1.14). The final match of a given query text against the *Subject* superclass is then calculated as the *best* match to either individual or composite subclasses. This change resulted in more effective performance on the broad range of subject queries. It introduced a small degradation in performance, but only a small one due to lazy evaluation of the match sets generated by the subclasses.

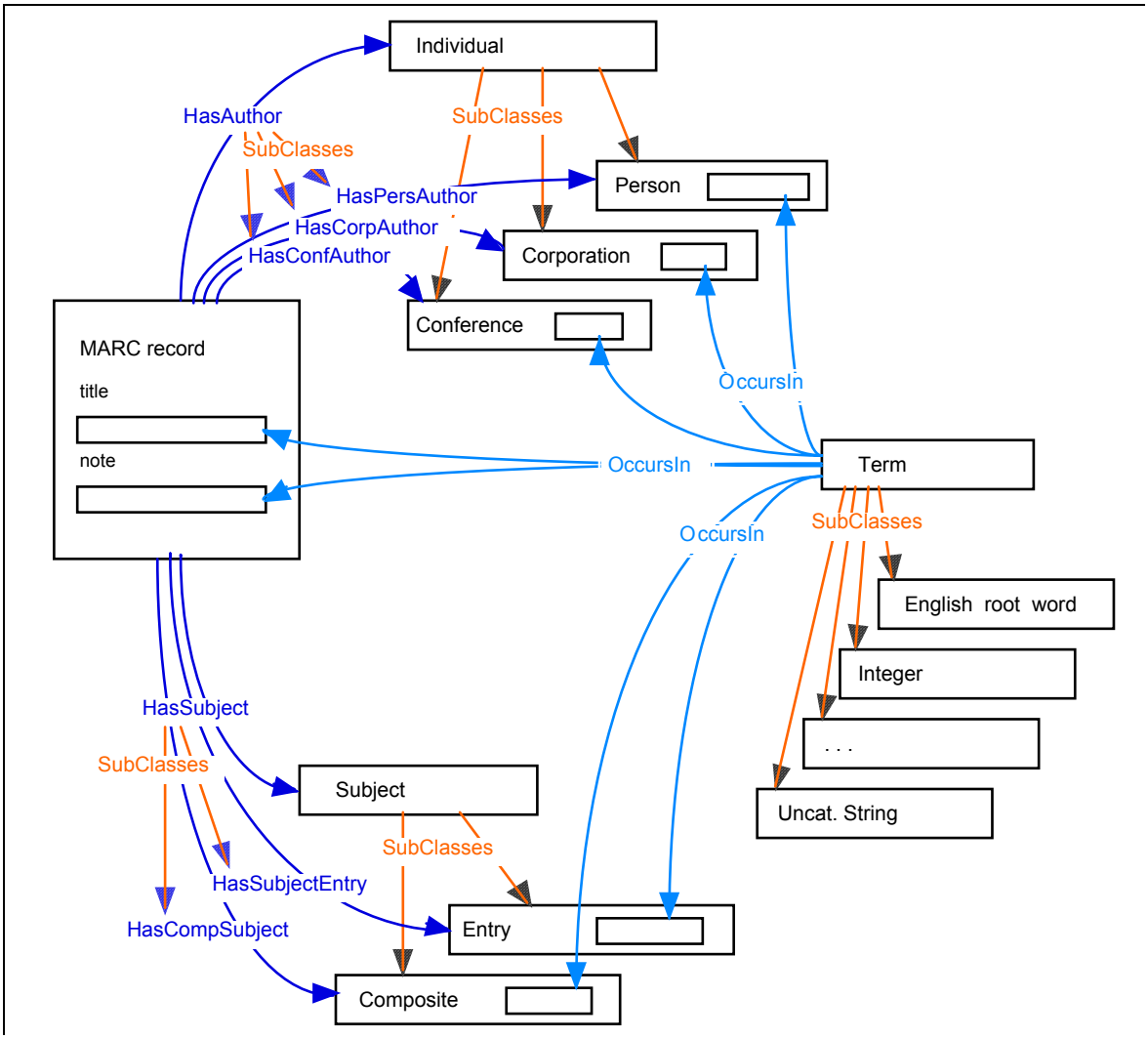


Fig. 1.14: The collection network for library catalog applications as used in the current version of MARIAN.

The class system depicted in Figure 1.14 is the one used by MARIAN at the time of this thesis for MARC collections. It supports all the query types discussed so far, as well as the multiple-access queries discussed in Chapter 2. With the addition of the fully-analyzed *Person, Corporation, Conference,* and *SubjectEntry* classes, it also supports browsing via "hot links" in the Web interface. When users access MARIAN via a Web browser, HTML pages are synthesized for works and groups of works at user request. As each work is presented, MARIAN links each author or subject entry to a query using the *Individual* or *SubjectEntry* object itself as a starting point. This provides a swift and sure way for a user to click from an interesting author or subject to all the works to which it is linked – a bonus only possible since the classes were materialized.

1.4 Summary

The stories and examples in this chapter illustrate the important features of the MARIAN system. Key representation features are nodes and links, use of classes to both organize data for easier design and exploit data regularities for better performance, and reliance on weights and weighted sets. Key operational features are lazy evaluation and reuse of common methods, especially common operations of search and similarity matching in semantically related classes.

The last discussion showed how change can come from many different directions and in service of many different needs. The splitting of the *Individual* class is an example of materializing existing structure: the distinction among the three types of individuals was already present in the data, and needed only to be reified. Generating redundant subject descriptions is an example of creating structure. There is an analogy here to data mining [Fayyad], although with data mining one is usually capitalizing in regularities in the data, while here we are working to suppress irregularities.

Both changes to the MARIAN library catalog collection are morphological changes. During the same period that they were made, the collection also changed through deletions, updates, and especially additions. In addition, the content of the objects in the node classes changed as we became increasingly sophisticated in our understanding of MARC records, what fields could be useful to retrieval, and how these should be treated. The total effect of all these changes is to make the collection dynamic in size, in content, and in both explicit and implicit structure. This is typical of digital library collections, and motivates our choice of an easily extensible and adaptable representational system. As we hope the preceding examples suggest, the combination of class hierarchies, node and link models, and weighted object sets provides a powerful set of tools for creating and continually modifying an effective information system.

The remainder of this thesis shows how all this is accomplished efficiently and effectively. Chapter 2 shows how all the searchers mentioned in this chapter can be built out of a very few components grounded in the theory of weighted object sets. Chapter 3 covers indexing while Chapter 4 explores stopping rules. Chapter 5 is adapted from a paper at ADL2000, giving details of MARIAN's use for electronic theses and dissertations. Chapter 6, adapted from a paper at ECDL2001, concludes this work, extending the discussion of Chapter 5, and emphasizing MARIAN's support for federated digital libraries.

1.5 References

[DCC] Dublin-Core-Community. 1999. The Dublin Core: A Simple Content Description Model for Electronic Resources. Dublin, Ohio: OCLC. http://purl.org/dc/

[Fayyad] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. 1996. Data Mining and Knowledge Discovery in Databases: An overview. Communications of the ACM 39(11)

[Fox&France] E. Fox and R. France. 1987. Architecture of an Expert System for Composite Document Analysis, Representation and Retrieval. International Journal of Approximate Reasoning 1(2): 151-175. Reprinted in "Readings in Information Retrieval", eds. Karen Sparck Jones & Peter Willett, pp. 400-412.

[Lassila] Ora Lassila and Ralph R. Swick. 1999. Resource Description Framework (RDF) Model and Syntax Specification. Cambridge, MA: W3C, Report REC-rdf-syntax-19990222, http://www.w3.org/TR/REC-rdf-syntax/

[Nelson] Michael L. Nelson, Kurt Maly, Mohammad Zubair, and Stewart N.T. Shen. 1999. SODA: Smart Objects, Dumb Archives. Proceedings of the Third European Conference on Digital Libraries, (ECDL'99, September 22-24, LNCS 1696). Paris: Springer, pp. 453ff. http://link.springer.de/link/service/series/0558/papers/1696/16960453.pdf

[Zimmermann] H.-J. Zimmermann. 1991. Fuzzy Set Theory and its Applications. Amsterdam: Kluwer.

Chapter 2: The MARIAN Searcher Community

[Searcher Design] [Searchers and Search Functions]

The MARIAN model of the universe is made up of classes of nodes and classes of links. Handling each class is a *class manager*. Class managers are responsible for all information objects of their class. New objects of the class are added by interacting with the class manager, as are alterations and deletions when these are permitted. Class managers map full-fledged information objects into FullIDs and back, and implement appropriate methods for determining object identity and equality. They can produce appropriate measures of the numbers and types of objects in the class. In general, they are responsible for determining and maintaining the extension of the class at any given moment in the lifetime of a collection. Some class managers – the *basic* class managers of a system – accomplish this by maintaining databases of class instances.

Between them, the basic class managers in a MARIAN-based digital library divide up their universe into disjoint sets of nodes and links, each the responsibility of a single manager. However, not all classes in MARIAN are disjoint. Both the node and link class systems are lattices, so any collection of link or node classes is guaranteed both a meet and a join. Meets are implemented using superclass managers, the extensions of which are defined as the union of the extensions of their subclasses. Superclass managers have no databases of their own, but rely on the existence of basic class managers somewhere below them in the lattice. Joins of basic classes are avoided in MARIAN. Instead, we insist that the basic classes be disjoint. In lattice-theoretic terms, their join is the absurd class, which has no members.

Among their other functions, each MARIAN class manager implements one or more search methods. A search method is a mapping from abstract object descriptions to weighted sets of objects matching that description, where the weight of each object in the set functions as a measure of how well that object matches the description. We call these sets *match sets*.

Any class in the MARIAN world view can support search methods particular to itself. As mentioned earlier, the *EnglishRoot* class supports a method which matches strings to sets of word roots that might produce that string through the transformations of English morphology. More generally, search methods are defined in MARIAN for *Text* and *StructuredDocument* node classes and for *HasAttribute* and *TermOccursInText* link classes. Search methods once defined may be inherited from superclass to subclass.

Two commonly used forms of searching are link activation and search within context. Link activation is the process whereby attention or focus on a set of nodes spreads to related nodes over intervening links. As an example, consider searching for library works with a certain person as author. In MARIAN's terms, this is a search for objects of class *Work* linked by one or more *HasAuthor* links to objects of class *Person*. The user puts together a description of a person. Identifying possible persons that match the description is a function of the *Person* class manager: the result of such a match is a weighted set of *Person* objects of decreasing similarity to the description. That set can be returned directly to the user, usually within some context for the *Persons:* books by them, books about them, an so forth. The user is more often interested, however, not in the persons themselves but in what they have written. Thus the weighted set of *Person* objects can also serve as input to a link class searching method, which returns a weighted

set of library works each linked to at least one person in the input set. In a more general case, link activation can proceed through arbitrary numbers of intermediate nodes and links before reaching objects of the desired class. In the most general case, entire patterns and topologies surrounding the desired objects may become activated. In this case, we say that the activated subgraph defines a context for the desired objects.

In a network of information objects, the context of an object is identified with its position in the rest of the network, which is expressed by the type and number of links between an object and its neighbors. Searching for an object in context thus becomes a process of searching through various link-node patterns in the network, optionally searching the object's own class with a node description, and then combining the intermediate results. Since each link-based search results in a weighted object set of nodes in the target class, as does any node-based search on the object class itself, the problem of assembling the context becomes one of weighted set algebra.

How efficient a searcher can be depends largely on what function it is implementing and how much information it has about its inputs. Some functions are inherently more difficult to compute than others. Some can be easy in certain contexts but difficult in others. And some are costly in the general case but can be implemented with much improved efficiency when a little extra information about their inputs is provided. We will present examples to illustrate these and other situations in this chapter.

2.1 Searchers for superclass managers

Consider, for example, the searcher for a superclass of disjoint subclasses. The function implemented is (weighted) disjoint union. No object in any of the sets being combined will appear in any other set, so no resolution function need be employed. The searcher need only take objects from (weighted) sets produced by its subclass managers, optionally scale them to promote certain subclasses over others, and return them in weight order (Fig. 2.1).

Disjoint union is an attractive operation in that it can work optimally with only the most local information about the incoming sets. Perfect performance with optimal efficiency is obtained when just the weight of the top element of each incoming set is known, and when the highest-weight element among those can be chosen with optimal efficiency. For k incoming proper weighted sets, optimal efficiency is $\boldsymbol{O}(\lg k)$, although we may chip away at this. Disjoint union of proper weighted sets is the most common case we have observed in superclass managers, so optimal efficiency here is worth striving for.

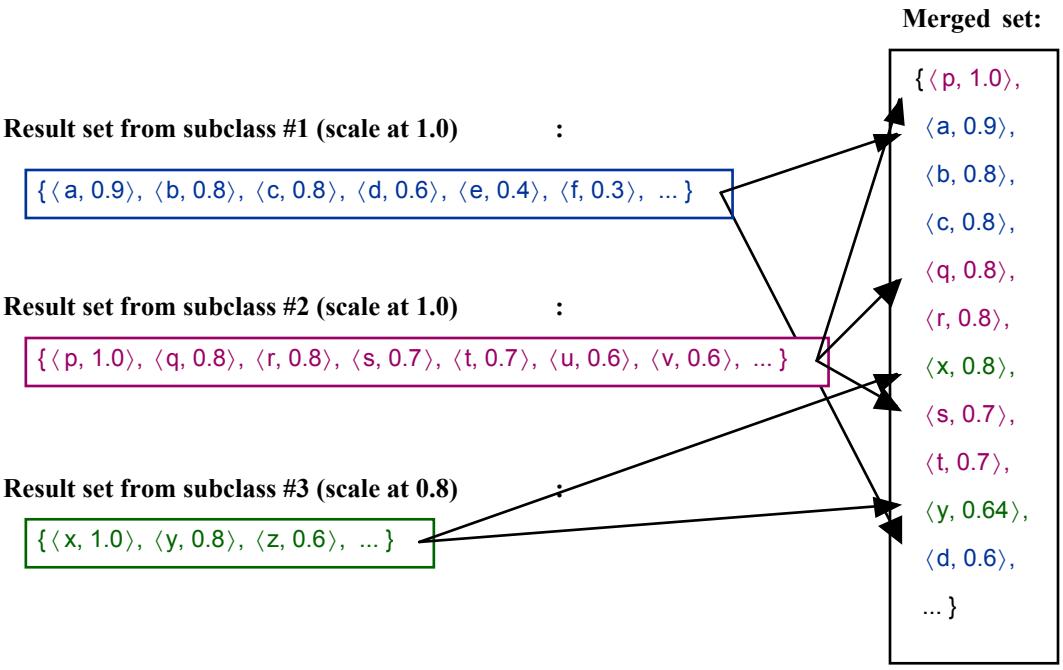


Fig. 2.1: A partitioned superclass search is implemented by weighted disjoint union of sets returned by subclass managers. The superclass manager scales the incoming sets and merges them into weight order. Here subclass #3 is discounted relative to the other subclasses, so elements x and y are down-weighted in the visible portion of the resulting merged set.

This analysis, though simple, is interesting because it provides a way to calculate the *cost of the class structure* in searching. Often in designing digital library representations we are faced with a choice between regarding a large group of information objects as making up a single class and regarding them as a union of subclasses. Chapter 1 discusses two examples of this choice. In all cases there are semantic advantages to be gained by partitioning the group into disjoint subclasses, and in some cases partitioning allows us to save computational costs on restricted searches in a single subclass, but in all cases partitioning adds an additional cost to the operation of searching the entire group. We can now measure this computational cost: it consists of a constant factor of $lg\ k$ for each search operation, where k is the number of disjoint subclasses. This cost is invariant with the size of the match set and is distributed evenly over the full process of developing the union set, both desirable factors. Further, it is small in comparison to the number of subclasses, which is itself generally a small number. Thus we say that a disjoint union representation has a relatively low cost, because a disjoint union searcher adds only a small constant to the full cost of the search operation. This is due to the high efficiency of the disjoint search operation.

Perfect efficiency can be achieved if all the subclass managers produce flat sets. In that case, all elements in each incoming set will have the same weight and can all be added to the union as they are produced by the subclass manager, with no additional operations required. Only when each set is exhausted does the superclass manager need to do any work, and then it is limited to determining the set with the next highest weight. When the number of subclasses is small compared to the overall size of a union set, this cost is vanishingly small. In fact, the cost is minute whenever the number of subclasses is small in an absolute sense – and as long as digital library class structures are built by humans rather than machines, the branching factor of a class

hierarchy will always be small in an absolute sense. Thus when we have a superclass of disjoint subclasses, each of which returns flat match sets, we can achieve efficiency so high that the class structure imposes no noticeable cost. This is not a common situation – the more common situation is proper weighted set – but it will serve us as a baseline against which to measure searcher efficiency.

Unfortunately, we cannot always count on superclasses dividing neatly into disjoint subclasses. The MARIAN class system is not a pure hierarchy, but a lattice where subclasses may themselves share subclasses. Thus MARIAN systems also require a general purpose superclass manager whose search function can handle incoming sets that may have elements in common. Such an *overlapping (or general) union searcher* must implement a resolution function to determine final weights for common elements. We could argue that the most natural function to model the class join function is the classical fuzzy set union operator, which resolves weights for common elements to the maximum incoming weight. *Max()* is a particularly happy function for weighted sets in the MARIAN realization because MARIAN's weighted sets are always produced in non-ascending weight order. This means that an element can always be returned the first time it is encountered, as that occurrence will always be one with highest weight. Once an element has been returned, all further occurrences of it can simply be ignored (Fig. 2.2).

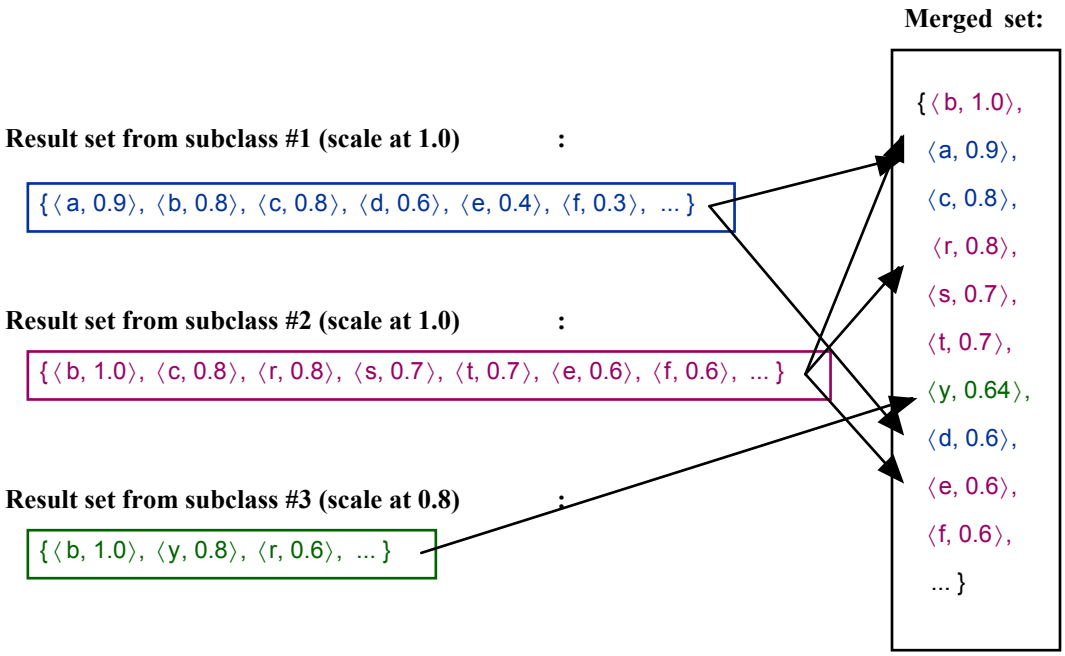


Fig. 2.2: The more general case of a superclass of (potentially) overlapping subclasses requires some mechanism for resolving items returned by more that one subclass manager (in this example, the objects b, c, e, f and r). The most common resolution function is to chose the maximal weight in any subclass.

Unfortunately, even ignoring duplicated elements is not a computationally trivial operation. A searcher can only ignore subsequent occurrences of an element if it can recognize them. There are at least two basic approaches to the problem: we can identify the duplicates ahead of time, or we can remember which elements we have seen. The first approach has an $\boldsymbol{O}(N_c \cdot lg\, N_c)$ cost

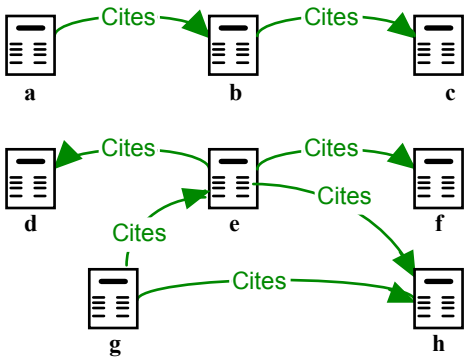
in the general case, where $N_c = \sum_i N_i$ is the total number of elements in all the incoming sets, including duplicates. The second requires the searcher to maintain a short-term memory or abstract table of elements seen to date, checking each new element against the table to tell whether or not it should be discarded. If the table is implemented using one of the classic "good" dynamic data structures such as a balanced tree, a probe of the table would cost $\boldsymbol{O}(lg\, N_T)$, where N_T is the current size of the table. In that case the operation of fully merging the component sets would have a total cost of $\boldsymbol{O}(N_c \cdot lg\, N_T)$ which is approximately the same as the first approach. There is a major difference between the two approaches, however. In the first, all the cost is "up front" – a constant regardless of how much of the result set is returned. In the second, the cost is distributed evenly over the process of realizing the set, at a rate of $\boldsymbol{O}(lg\, N_T)$ per element returned. This makes the second approach a clear winner for lazy systems like MARIAN.

In practice, we can do better than $\boldsymbol{O}(lg\, N_T)$ per table probe by using a well-tuned hash table to store the elements seen. The same dodge would improve the first approach equally, so the table implementation has no effect on the preceding discussion, but it does have an effect on searcher performance, so we will make a note of it here and return to it in the next chapter.

2.2 Searchers for link managers

Link class managers in MARIAN all support two basic search methods: node activation and weighted set activation. Both methods are based on the definition of link identity: two links are the same if they belong to the same class and if they link the same two nodes in the same direction. Recall that all links in MARIAN are directed links: they run from a source node to a sink node. For two links to be identical, or for a link description in a query to match a network link in a collection, the links must run in the same direction: to say that p is the author of w is a very different assertion than that w is the author of p.

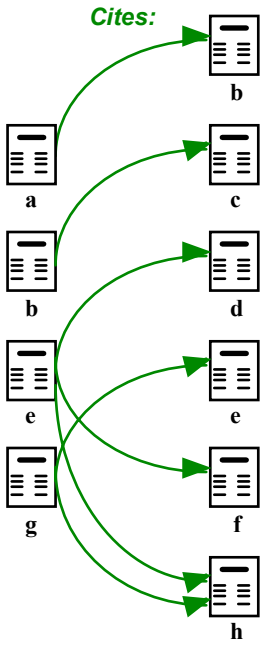
Node-based activation is the basic function of the link class manager: given a *key* node and the information whether that node is to play the role of source or sink node, return all *target* nodes that are exactly one link of this class away from it in the correct direction. If we envision all the links in this class as running in a single spatial direction, say from left to right, node activation can be thought of as finding all the nodes on the target side of the space that are linked to a particular node on the key side, where either source or sink can be either key or target (Fig. 2.3). This is a set-valued operation, since many nodes may be linked to the key node by links in this class and in the correct direction. What is more, if this is a weighted link class, node activation is a weighted-set-valued operation. This brings up a question: if a target node is activated by a weighted link from a weighted key node, what is the final value for the activation?



(a)

```
links(Cites, a, SOURCE_TO_SINKS, {b})
links(Cites, a, SINK_TO_SOURCES, ∅)
links(Cites, b, SOURCE_TO_SINKS, {c})
links(Cites, b, SINK_TO_SOURCES, {a})
...
links(Cites, e, SOURCE_TO_SINKS, {d, f, h})
links(Cites, e, SINK_TO_SOURCES, {g})
...
links(Cites, h, SOURCE_TO_SINKS, ∅)
links(Cites, h, SINK_TO_SOURCES, {e, g})
```

(b)



(c)

Fig. 2.3: **(a)** A portion of a citation network among technical papers, represented as an object-level graph. **(b)** The operation of node activation can return a single node, a set of nodes, or no nodes depending on the direction of the link. **(c)** The *Cites* link space can be envisioned with all links running from left to right. Note that papers **b** and **e** occur on both sides of the link space.

Any network representation system that supports the concept of link activation must provide a way of calculating how strongly nodes are to be activated. Many neural network systems provide attenuation functions, assuring that the spreading pattern of activation from an originally excited node will die out unless otherwise reinforced. In MARIAN, we have not found this to be necessary. On the other hand, we do need to give a mathematical expression to the effect of a weighted link on a level of activation. If *Person p* is activated at weight ω (representing perhaps that p has probability ω of being the person described by a user's query) and *p* is linked to *Work q* by a link with weight ξ, what weight should be assign to *q?* As with resolution functions, a number of mathematical functions are reasonable for this *scaling* operation: the simplest is the algebraic product $\omega \cdot \xi$.

The second basic link class search function is weighted set activation: given a weighted set of nodes occurring on the key side of the link space, return all nodes of the target side that are directly linked to any member of the incoming set. Since some nodes on the target side may be linked to more than one key node, a resolution function is essential in this class, just as in the general superclass manager. An effective function that captures the semantics of digital library links like *HasAuthor* and *HasSubject* is again the simple maximum. The [internal structure / efficient implementation] of link searchers differs from superclass searchers, however, because the information available is different.

Consider first the case of an unweighted link class. In such a class, a link either exists or it does not, so all existing links can be regarded as having `topWt`. Put another way, if the key node

arrives with a weight ω, then we can assign the same weight to the target node.[1] Thus the only factor determining the weight and order of target nodes in the result set is the weight of key nodes. As in the overlapping superclass searcher, we can implement a maximizing resolution function by the simple expedient of adding a target node to the result set the first (and only the first) time it is encountered. As before, assuring this discipline requires some sort of a table, though only one that permits recognition of elements already seen. The difference with this searcher is in how the results are developed.

In contrast to the searcher for a general superclass, an unweighted link searcher is already presented its keys in weight order. Thus it can develop its result set by the simple expedient of taking a key node, finding all links attached to that node in the correct direction, checking whether the nodes on the other end have been seen before, and adding any new ones to the result set (Fig. 2.4). The information available is already enough to determine the order of the result set, so *N* elements can be produced with ***O****(N)* effort.

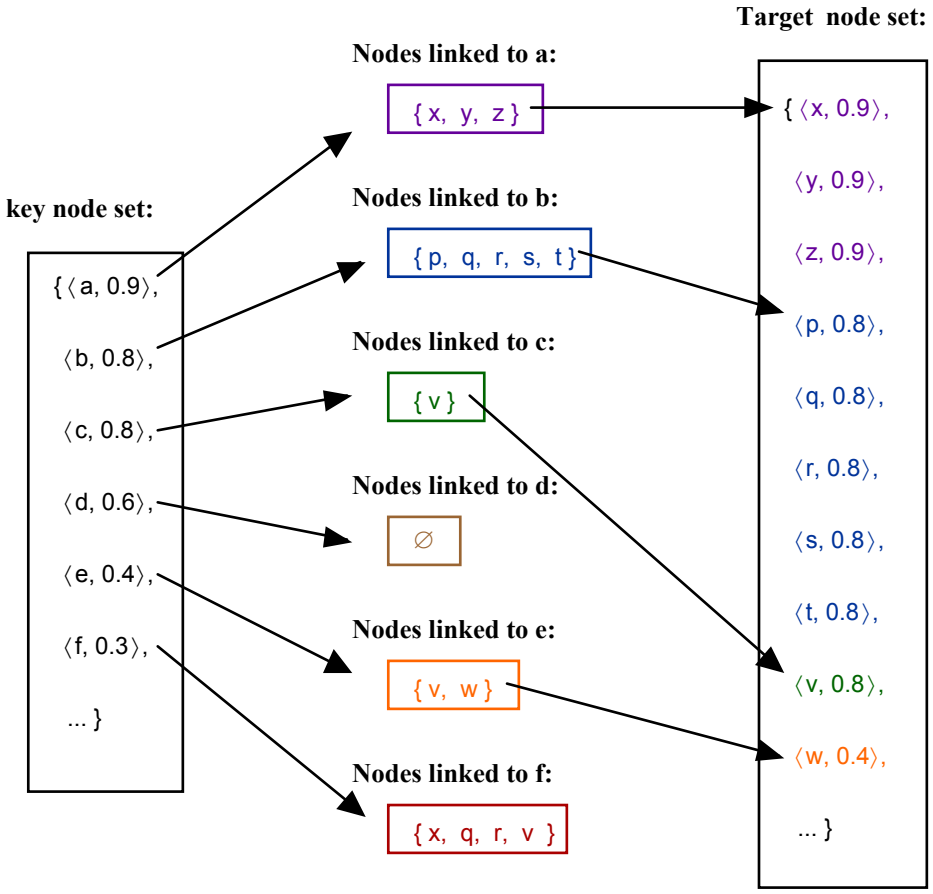


Fig. 2.4: Developing a link activation set from a class of unweighted links. The target set is constructed of nodes linked to each of the key set nodes in turn, with weights copied from the key weights. Duplicates at lower key weights are discarded, for example node v in the set { v, w } and all the nodes in { x, q, r, v }.

[1] This is the system actually used in MARIAN, corresponding to perfectly trustworthy links and an attenuation factor of zero. A non-zero attenuation factor would result in final weights that are uniformly lower than the incoming weight, as would activation of links from a class whose weight was uniform but less that topWt. Both these additions are options in the formal model, and both are covered by the same searcher operation, so we will confine our exposition to the simplest.

Weighted link classes require a more sophisticated searcher. The situation parallels the move from a partitioned to a general superclass. Not only must duplicates be recognized and discarded, but the next most highly weighted element can be found in any of several sets (Fig. 2.5). In the case of a superclass of overlapping subclasses, the incoming weighted sets were the match sets generated by the subclass managers; here, the incoming sets are the weighted sets of nodes adjacent to each node in the key set. The difference in the two situations lies in the numbers of sets involved. The class structures in MARIAN systems are typically created by human beings, and thus are built to a human scale. Typical superclasses involve three, or five, or at most a dozen subclasses. The key node sets presented to link searchers, in contrast, can be as large as any search result set in the system: often thousands or tens of thousands of nodes in size.

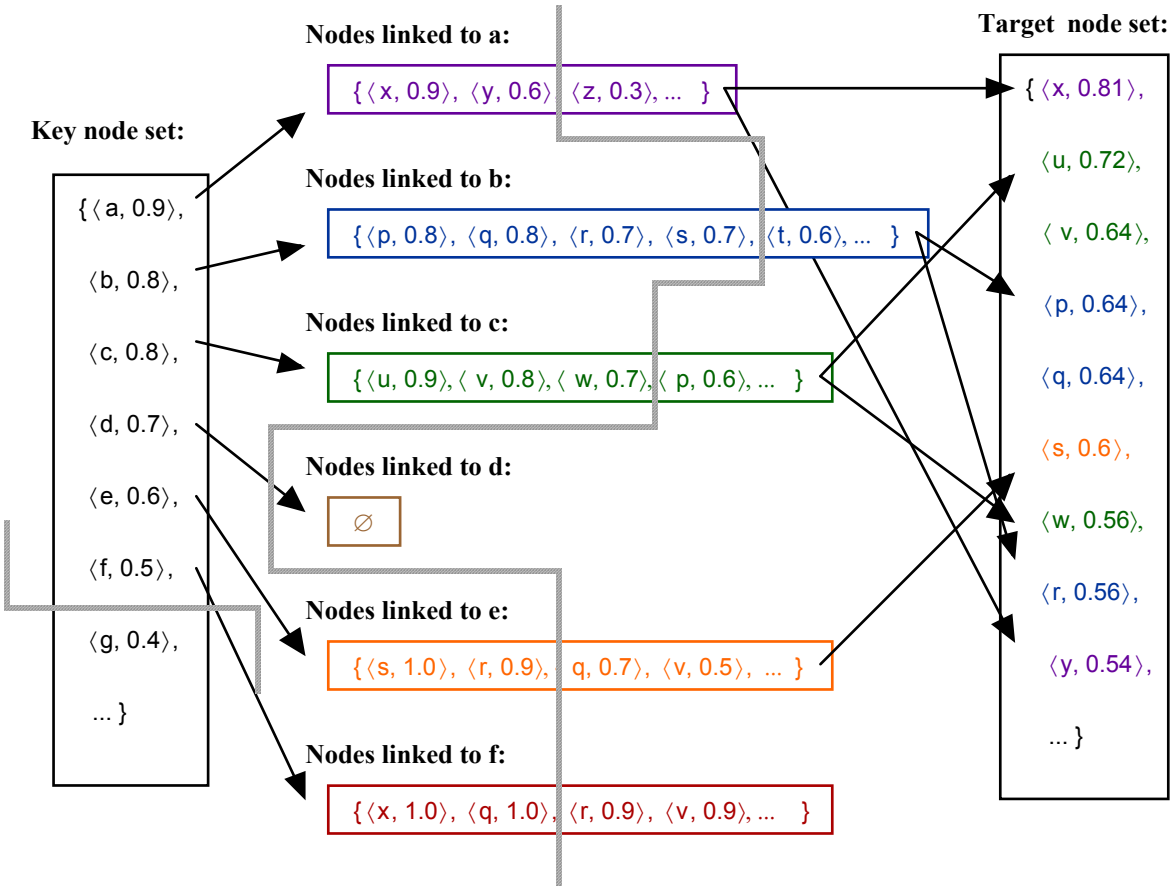


Fig. 2.5: Developing a link activation set from a class of weighted links is a more complex process, as result nodes' weights are determined by both the weight of the key node and the weight of the link. Pictured here is a result set developed down to weight = 0.5. Frontiers, shown by textured lines, demarcate how far each set has been explored. As can be seen in the result set, which link set will supply the next element is not predictable beyond the frontier.

Lazy evaluation serves us well here. At any point in the development of the result set, it is possible to construct a frontier through both the key node set and the link sets. Elements above the frontier have been explored. In the case of the key set, their link sets have been discovered and "added to the mix." In the case of link sets, their elements have been either added to the

result set or discarded. Elements below the frontier have not been explored and do not need to be until the result set is further developed. In the case of the link sets, the elements below the frontier have combined weights less than (or equal to) the last result set element developed. In the case of the key set, nodes below the frontier have initial weight so low that all of the target nodes they are linked to – even those whose link strength is `topWt` – will have a combined weight less than or equal to the last developed element in the result set.

Since all the sets are presented in weight order, only those elements immediately below the frontier are candidates to be the next element in the result set. Development of the result set can thus proceed lazily in the sense that only a well-defined and relatively small set of elements – the elements along the frontier – must be examined in order to determine the next result element. Unfortunately, this set is only small relative to the full size of the incoming sets. Depending on the drop-off functions of the key set and link sets, it is possible for the frontier to be as large as the size of the key set. It is also possible that it might include only an element or two, in the case where the key set has a precipitous drop-off function and the link sets are relatively flat and even. In the next chapter, once we have a statistical model of the sorts of drop-off functions expected in digital library applications, we can make a better estimate of the expected cost of working through the frontier. In the meantime, we note only that even the worst case is $\boldsymbol{O}(N_{Key})$, where an exhaustive approach would be $\boldsymbol{O}\left(\sum_i N_{LinkSet_i}\right)$ or approximately $\boldsymbol{O}(N_{Key} \cdot N_{LinkAvg})$, where N_{Key} is the size of the key set, $N_{LinkSet_i}$ is the number of nodes linked to key node *i*, and $N_{LinkAvg}$ the average size of a link set for that node class, link class and direction.

2.3 Searcher internals: sequencers and tables

At this point, some common substructures of the searcher class are becoming apparent. In particular, all of the searchers described so far depend heavily on *sequencers* and *tables*. This will continue to be true for more complex MARIAN searchers. Tables have been mentioned in the last two sections, so we will begin with sequencers.

A sequencer is a software object that takes a set of incoming streams of weighted objects in weight order and produces a single output stream also in weight order. Typically though not necessarily the input streams are presentations (in Java, *Enumerations)* of a group of weighted sets. The output stream consists of exactly the same elements with exactly the same weights as in the input streams, shuffled into a single non-increasing order by weight. Duplicates are not removed.

MARIAN sequencers, like MARIAN weighted sets, are designed and implemented for lazy evaluation. They do not immediately develop the output stream when created, but instead provide their clients a presentation of the final sequence. The presentation is developed only as far as needed. Thus we describe sequencer behavior with conditionals: if the sequencer has returned elements down to weight ω, then elements of weight >ω in any input set have been presented; if the sequencer has completed its output stream, then it has returned $\sum_i N_i$ elements, where N_i is the number of elements in input set *i*.

Java MARIAN uses several different sequencers, all with the abstract behavior described above. Internal differences are determined by two practical questions: how many input sets are expected, and how they are discovered and added to the incoming mix. Sequencers that handle a large number of input sets require sophisticated data structures to efficiently find the next element to output; those with only a few input sets achieve greater effective speed with a linear structure. Some sequencers can be presented with their input set collection when they are created; others – most notably those used in link searchers – need to develop the input sets dynamically from a (weighted) set of input keys.

We can regard a sequencer as being made up of a priority queue of "active" weighted set presentations together with a control module (Fig. 2.6). The priority queue holds an indefinite number of set presentations – or more generally, presentations of any ordered sequence of elements – in order by the weight of the top element in each. The control unit serves as gatekeeper for presentations going into the queue and elements coming out.

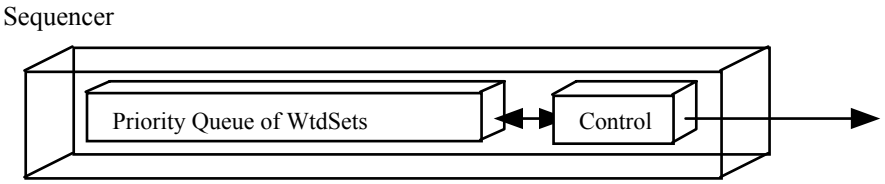


Fig. 2.6: A Sequencer consists of a priority queue of weighted set presentations together with an input/output control unit.

The weighted set priority queue has only three significant methods: add a new presentation, remove the top element of any presentation in the queue, and report the top weight in the queue. Two implementations of this construct are used in MARIAN (Fig. 2.7). The LinearWtdSetQueue maintains the active presentations in an unordered vector and uses linear search to establish the top element each time one is requested. The HeapWtdSetQueue uses the standard heap implementation of a priority queue to provide $O(\lg k)$ access to the next element.

Both implementations make use of a simple refinement to increase efficiency. We observe that most actual weighted sets observed in running information systems do not have continuous drop-off functions, but "stair-step" functions where several consecutive elements have the same weight. Thus the chances are good that at any point the next element to be returned will come from the same set as the current top element. Based on that observation, both priority queues keep references to the top two set presentations. The LinearWtdSetQueue finds the two top set presentations when it is created, and thereafter does no searches until the top set falls below the second, when it promotes the second to first and makes a sweep through the vector to find a new second. The heap queue reserves the top set presentation outside of the heap structure until its top remaining element falls below the top element of any presentation still in the heap. Then it switches the reserved set for the set on top of the heap and reheapifies. For both queues, testing before searching institutes a cost of one additional comparison per element returned. However, it minimizes the number of times the linear sweep on the one hand and reheapification on the other need to be performed. This trade-off can prove a significant win.

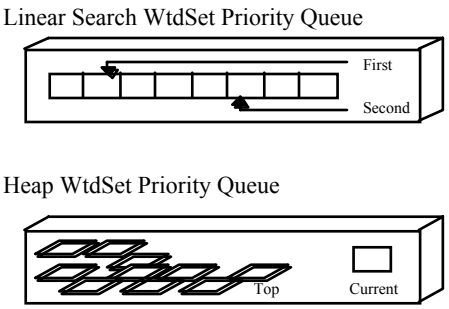


Fig. 2.7: Depending on the number of sets involved, sequencers use either a linear or a heap implementation of weighted set priority queues.

Both priority queue implementations provide the same functionality. The decision of which to use depends on the number of presentations expected, and on the well-known observation that linear search outperforms more sophisticated data structures and algorithms when the number of items to search is small (the general rule of thumb is that the watershed is around 20). The two implementations, and indeed the choice of which to use, is hidden from the client. Instead, the client provides an estimated number of sets when constructing the sequencer, and one of the two is chosen automatically based on that number.

The control units used in building sequencers are all lazy. They react to requests for sequenced elements by returning the top element of the queue as long as the queue is non-empty. Control units differ in how sets to be sequenced are presented to them, and thus in when incoming sets are added to the queue. We distinguish two cases, informing two sorts of sequencers.

In the case of the SetCollectionSequencer, incoming sets are added to the sequencer individually. The expected scenario is that all incoming sets will be added before any elements are requested from the sequencer, but nothing precludes adding additional sets during sequencing operation. This sequencer uses a control that adds each set to the priority queue as soon as it arrives (Fig. 2.8). This is the sort of sequencer used in superclass searchers, generally with a linear search weighted set priority queue.

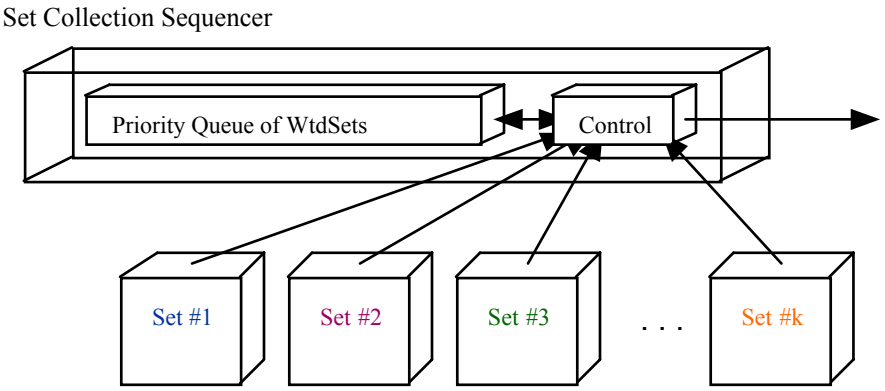


Fig. 2.8: In a SetCollectionSequencer, sets are added individually and explicitly. The control unit passes them immediately to the priority queue, then returns top elements off the queue until it is empty.

Control is more sophisticated in the SetMappingSequencer. This sequencer has the defining characteristic that incoming sets are determined by some "key" set together with a function that turns each key element into a further weighted set (Fig. 2.9). Set mapping sequencers are used in the weighted link searcher, for instance, where each key node is mapped into a weighted set of linked target nodes. The SetMappingSequencer uses lazy evaluation to minimize the costs of both developing the key set and set mapping. It accomplishes this by only adding new sets to the queue when the top element in the queue has lower weight than the top element in the key set presentation, or when the queue is exhausted but the key set presentation is not. As long as one of these criteria is met, the control unit takes an element off the key set presentation, maps the object part into a weighted set, then scales the incoming set by the weight of the object and adds the result to the queue. Only when the top element in the key presentation has lower weight than the top element in the queue does the control pass an element to the client, and only when both queue and key set are exhausted does the control acknowledge the sequencer as empty.

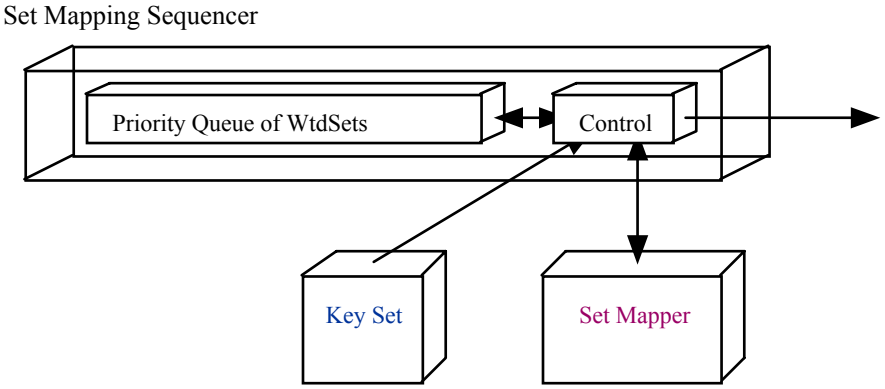


Fig. 2.9: A SetMappingSequencer makes use of a weighted key set and a function that turns each key element into a further weighted set. Here the control unit must regulate development of the key set so that incoming sets are only added to the queue as needed.

Tables are used in MARIAN searchers to keep track of those elements already seen. In the searchers we have seen so far, the elements in the tables carry no information beyond their very existence – either they are in the table, and should thus be ignored, or they are not, in which case they are new and may be returned. More complex searchers store additional information in the tables: a weight for the element, for instance, or the next element in some sequence. The table operations, however, are the same in all cases: add an element to the table, and test whether an object is in the table, returning any associated information if it is.

To test whether an object is in any data structure requires a way of determining object identity. This is not always a simple question: in Java, for instance, one can have two `Integer` objects both with the value 23. Whether these objects are regarded as identical or not depends on whether we are looking at them as numbers or as software, and whether they are evaluated as identical in a Java program depends on whether one uses the == operator or the `Integer.isEqual()` method. With digital library objects, identity questions become more complicated still. In the case of MARIAN information objects, identity criteria are determined by the object's class and final arbitration is left to the class manager. Under normal circumstances, fortunately, all objects input to a searcher have already been resolved into

`FullIDs`, so their criterion for identity is just whether they match in both class and instance ID. This is the criterion used in all MARIAN v. 2.1 searcher tables.

All tables in MARIAN v. 2.1 are thus *FullID tables:* abstract tables indexed by `FullIDs`. In Java, these are implemented using the utility library `HashTable` class, which provides ***O****(1)* access to objects that support a well-behaved hash function. Defining such a function turns out to be relatively easy for `FullIDs` (see class `FullID`). Java `HashTables` support the methods `put(Object key, Object data)` which adds the data to the table at the abstract location determined by the key, and `get(Object key)`, which returns the data object added by `put()` if the key is in the table and `null` if not. Some `FullIDTables` provide other means to access a table location, so the data `Object` in a `FullIDTable` always contains at least the value of the key.

Most of the `FullIDTables` we use to construct `Searchers` are actually *threaded* (in the linked list sense, not the process sense). In other words, in addition to whatever objects are stored as data, the hash table cells also include either single or double links, allowing us to connect them into orders that have nothing to do with the table construction. In a `SinglyThreadedFullIDTable`, the objects in the table form a singly linked list (Fig. 2.10). A `DoublyThreadedFullIDTable` includes both "prev" and "next" references, facilitating insertions and deletions within the thread. The `DoublyThreadedFullIDTable` and its subclasses will be discussed below.

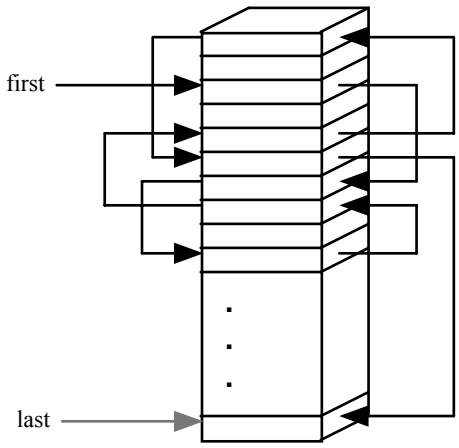


Fig. 2.10: A singly-threaded full ID table includes a sequence of objects, each of which references the next in some order. Typically, this order relates to the weights of the objects, but has nothing to do with the natural order of the table or the order of the key objects used to reference individual cells.

2.4 Searcher anatomy: superclass and link searchers

We are now in a position to examine the internal anatomy of the searchers we have met thus far. The early sections of this chapter discuss four class managers, each of which functions as a Searcher for its class: the `PartitionedSuperclassSearcher`, the `General-SuperclassSearcher`, the `FlatLinkSearcher` and the `WeightedLinkSearcher`.

From the viewpoint of weighted set algebra, however, we have discussed only two: the `DisjointUnionSearcher` and the `MaximizingUnionSearcher`.

The `DisjointUnionSearcher` is used only for the `PartitionedSuperclassManager`. The weighted set operation of disjoint union has the distinguishing characteristic that it requires no resolution function. As a consequence the `DisjointUnionSearcher` requires no table structure, but consists only of a `Sequencer` (Fig. 2.11). Since all `Sequencers` have the same functional profile – in Java terms, since they implement the same interface – we need not specify what sort of `Sequencer` to use, or how the incoming sets are to be added (see class `DisjointUnionSearcher`).

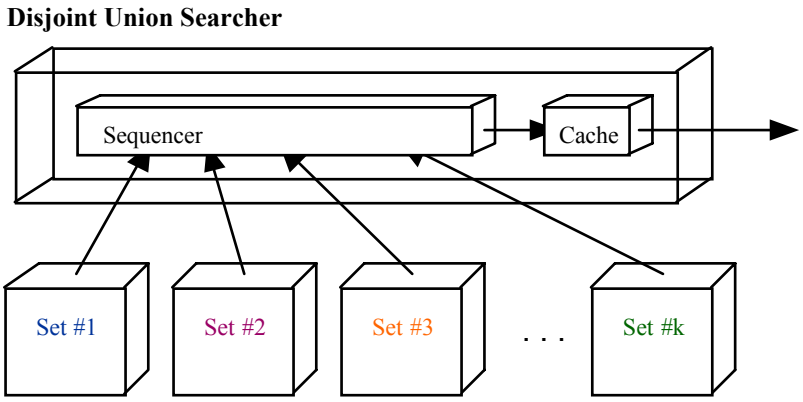


Fig. 2.11: The DisjointUnionSearcher object is simply a Sequencer on the incoming sets together with a simple cache to support MARIAN WtdSet behavior. The type of sequencer depends on the number and genesis of the incoming sets.

One further refinement to the `DisjointUnionSearcher` stems from our requirement that all `Searchers` behave as weighted sets: in this case, weighted union sets. A `WtdSet` object in MARIAN spawns lazy presentations; in Java, particular sorts of `Enumerations`. Any `WtdSet` object may be used by multiple clients, which it satisfies by passing a presentation to each. Efficiency is thus best served by developing the set as far as called for by the most extensive presentation and (under most circumstances) by caching the sequence of elements developed so that other presentations can follow it without duplicating effort. In the case of the `DisjointUnionSearcher`, a cache is added in the form of a simple vector or linked list of `WtdObjs`. The union set produced by the `Sequencer` is stored in the cache, and all presentations are directed through the cache. When any client of the `Searcher` asks for a presentation to be extended beyond the contents of the cache, the `Sequencer` is invoked to determine the next element. Only when the `Sequencer` has run through all elements of all input sets does any presentation report the end of the union set. At that point, all incoming elements are in the cache, and the `Sequencer` can be discarded together with the presentations it was using to explore the input sets.

A `MaximizingUnionSearcher` uses maximum() as its resolution function. As noted above, this function is easily implemented in the MARIAN environment by returning the first occurrence of any element in a merged sequence of incoming elements and ignoring subsequent occurrences. As with the `DisjointUnionSearcher`, we must allow for presentation of the

resultant set to multiple clients. This can be done using the same mechanism as before (Fig. 2.12), but the mechanism actually used in MARIAN is slightly more efficient.

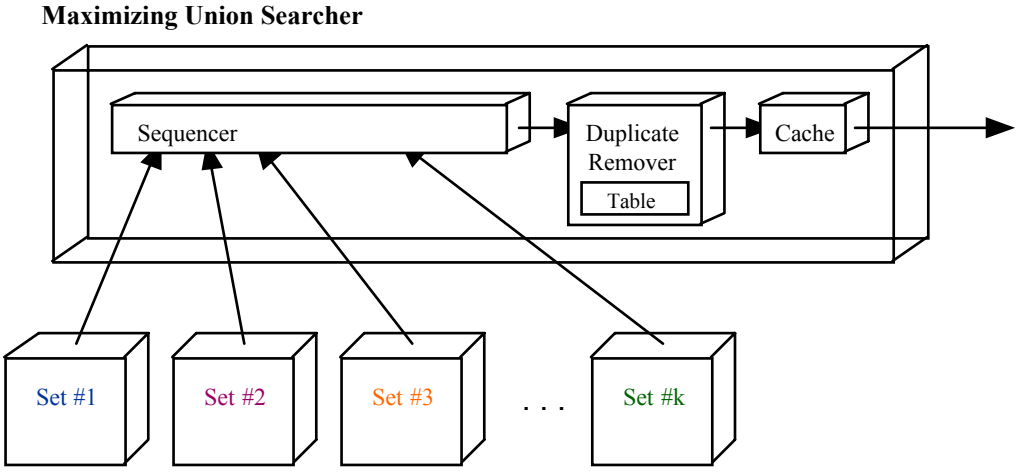


Fig. 2.12: A straightforward implementation of the maximizing union operation would require a basic FullIDTable so that duplicate elements could be recognized and discarded, followed by a cache to facilitate sharing the union set among multiple clients.

Recognizing duplicates requires only the most basic table structure: one capable of storing the key object itself and reporting whether or not the key is already in the table. Tables are more versatile than this, however. For instance, we can store at each cell in the table the key object, its weight when first seen, and a reference to another cell in the table. This forms a *singly threaded weighted object table,* which is nothing more than a singly threaded FullID table with weights added to each cell. Following the thread from first cell to last produces a sequence of weighted objects. If the objects are added to the table in the order they arrive, this table (now called an `InsertionOrderedWtdObjTable`) provides both a check for duplicates and an image of the maximizing union set produced.

The MARIAN maximizing union searcher uses a threaded table in just this way (Fig. 2.13). As with the `DisjointUnionSearcher`, a `MaximizingUnionSearcher` can be formed around any sort of Sequencer. When the searcher is created, a new `InsertionOrderedWtdObjTable` is created and the first weighted object in sequence loaded into it. Any presentation to a client includes a reference to this first object as the first object in the union set. As the presentation is extended, the thread is followed until it reaches the last object in the thread. At this point all objects in the table have been presented, but due to the lazy evaluation of the union more objects may still exist in the incoming sets. Therefore, when the last object in the table has been presented, more elements are requested from the sequencer until either a new element is found and placed in the table – in which case it has also been added to the thread and can be presented to the client – or the sequencer runs out of elements. Only when the sequencer has passed along all the elements in all the incoming sets is the presentation at an end. At that point, the sequencer can be discarded and the threaded sequence within the table used as the cached union set.

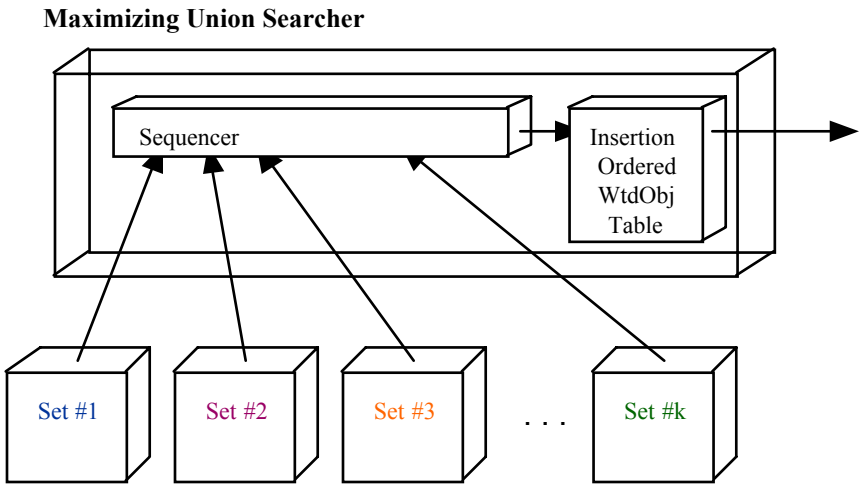


Fig. 2.13: As the MaximizingUnionSearcher is actually implemented in MARIAN, caching and duplicate removal are both accomplished by the same object, a singly threaded table of weighted objects maintained in insertion order.

The two superclass searcher objects use these two union searchers in parallel ways. The `PartitionedSuperclassSearcher` uses a `DisjointUnionSearcher` and the `GeneralSuperclassSearcher` uses a `MaximizingUnionSearcher`, both built from set collection sequencers since the collection of incoming sets are known in advance. When a superclass manager receives a query, it passes the query to each of its subclass managers. Each subclass manager passes back a match set of subclass objects weighted by how well they match the query. The superclass manager optionally scales each set by the reputability of the subclass and adds them to a new `SetCollectionSequencer`, from which it creates either a `DisjointUnionSearcher` or a `MaximizingUnionSearcher` (Fig. 2.14). The searcher functions as the match set for the original query, and passes presentations to the original caller and any additional interested clients.

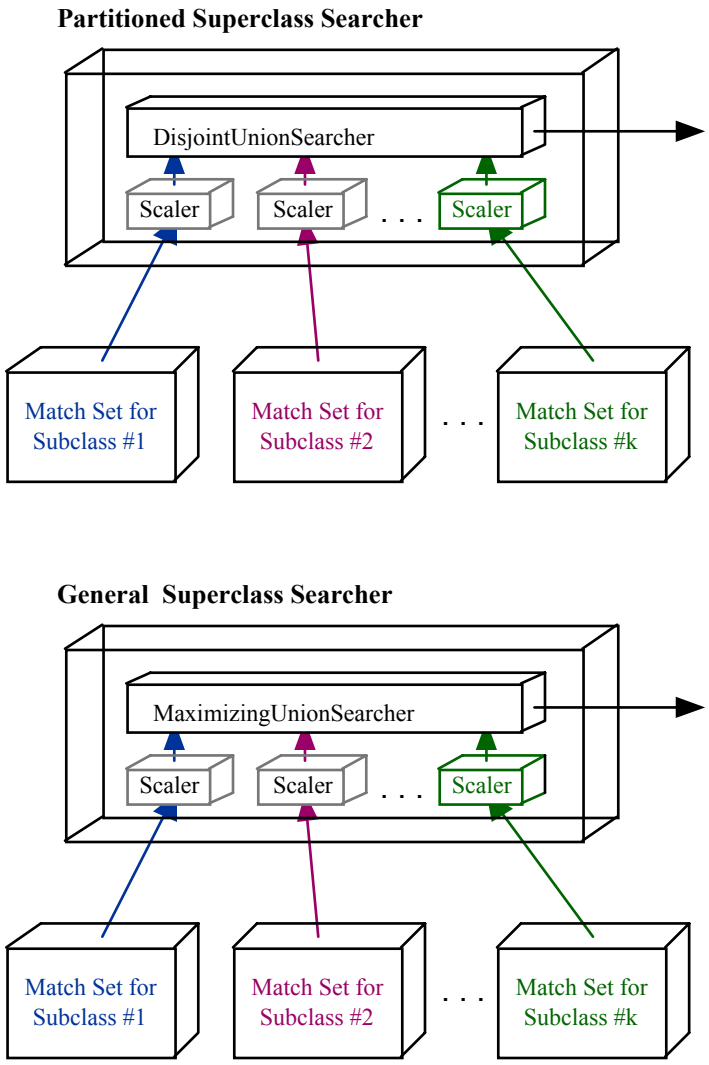


Fig. 2.14: The two sorts of superclass managers use different sorts of set operations, but in both cases return scaled unions of the match sets produced by their subclass managers. In this figure, scalers for Match Sets #1 and 2 are shown grayed out, corresponding to Figures 2.11 and 2.2, where subclasses #1 and 2 are both weighted at `topWt`.

Link searchers also use `MaximizingUnionSearcher` objects, but they use different sequencers to construct them. The `WeightedLinkSearcher` uses a `MaximizingUnionSearcher` constructed from a `SetMappingSequencer` (Fig. 2.15), where the mapping function is node activation: from a node on one side of the link space (an element in the key set), return a weighted set of all nodes on the other side. All the relevant differences between the operations of weighted set activation and general superclass searching – set mapping versus set collection, and most likely heap-based versus linear search – are hidden in the construction of the sequencer. The `FlatLinkSearcher` uses a truncated set mapping sequencer consisting of only a control module and no priority queue (Fig. 2.16). Its operation is simplicity itself: when a presentation reaches the end of the searcher table, the control unit takes the next element off the key set, maps it into an unweighted set of objects, and adds each object

to the table with the weight of the key element.[2] In both searchers, the result set is developed as needed, and only fully developed when both table and sequencer are exhausted, which is the default algorithm for a maximizing union searcher.

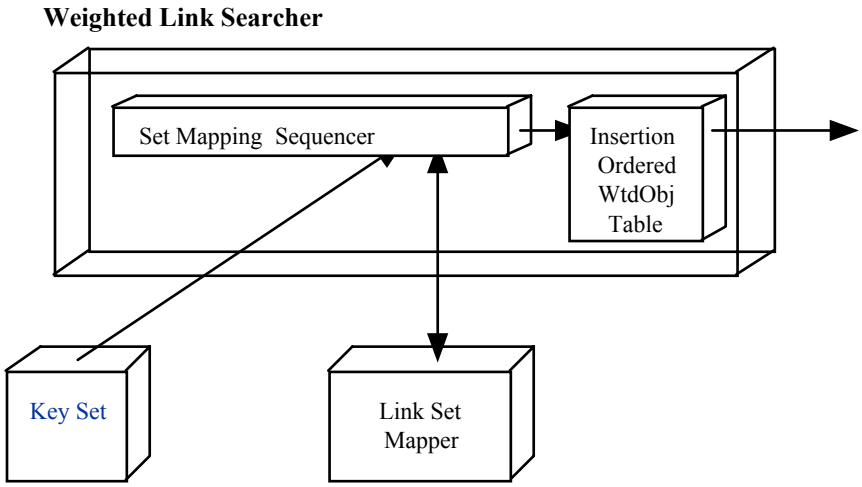


Fig. 2.15: The weighted link searcher is just a maximizing union searcher (here shown with its internal structure exposed) built from a set mapping sequencer rather than a set collection sequencer.

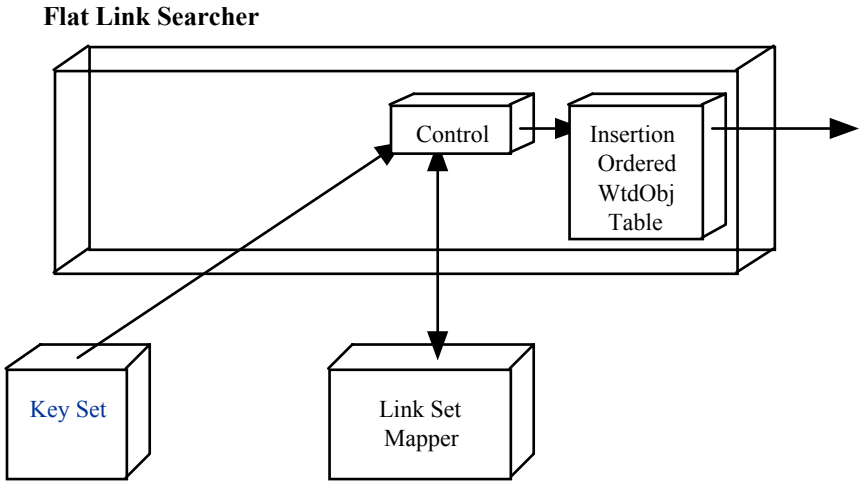


Fig. 2.16: The flat link searcher is a maximizing union searcher built from a truncated set mapping sequencer for flat sets.

2.5 Summative union searchers

The next level of complexity in weighted set combination occurs with the summative union searchers. A wide range of useful resolution functions all have the form

[2] It is at this stage that non-zero attenuation and uniform weights other than topWt would be treated, probably as scaling factors.

$$r(\omega_1, \omega_2, \ldots, \omega_k) = g(\sum_i f(\omega_i))$$

for some pair $\langle f, g \rangle$ of weight-valued functions. These are the summative resolution functions. Set union operations that use them are summative union operations, and a searcher that implements such a union is a summative union searcher. Examples include average and weighed average searchers, Euclidean distance searchers, and probabilistic sum searchers. The vector cosine similarity function used in SMART and related search systems is also a summative resolution function, although not an associative one, as is the relative entropy function used in the MARIAN text searcher.

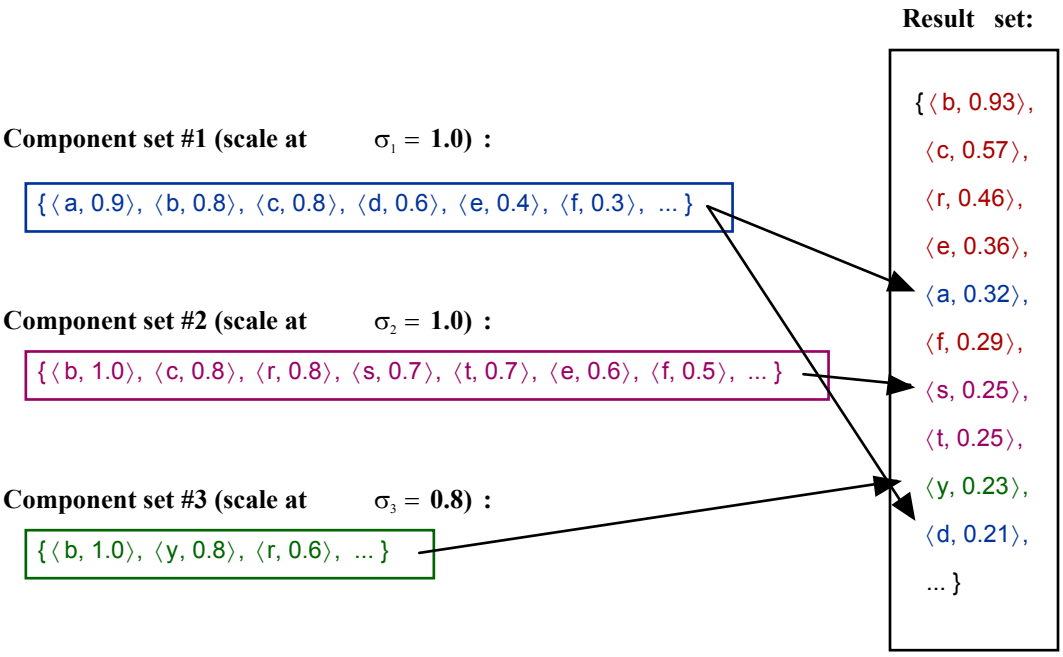


Fig. 2.17: In a summative union, the final weights are functions of all incoming weights. In this example, the function is weighted mean. Note that elements occurring in more than one set (colored brick red in the result set) tend to cluster at the top, but that elements with high weights in a single set (a, s, t, y, d, ...) may be interspersed.

Summative union searchers tend to promote elements that occur in several incoming sets (Fig. 2.17). Depending on the exact $f()$ and $g()$ chosen, this effect can be stronger or more subtle. The simple weighted average:

$$f(\omega_i) = \sigma_i \cdot \omega_i \qquad g(\omega_t) = \omega_t / \sum_i \sigma_i$$

shown in Figure 2.17, for instance, tends to promote common elements more often than a weighted sum-of-squares:

$$f(\omega_i) = \sigma_i \cdot \omega_i^2 \qquad g(\omega_t) = \omega_t / \sum_i \sigma_i$$

or normalized Euclidean distance:

$$f(\omega_i) = (\sigma_i \cdot \omega_i)^2 \qquad g(\omega_t) = \sqrt{\omega_t} \Big/ \sqrt{\sum_i \sigma_i^2}$$

since both of the latter give a high premium to single large values (Fig. 2.18). All other things being equal, however, any summative function will promote elements that occur in more sets above elements that occur in fewer at similar weights.

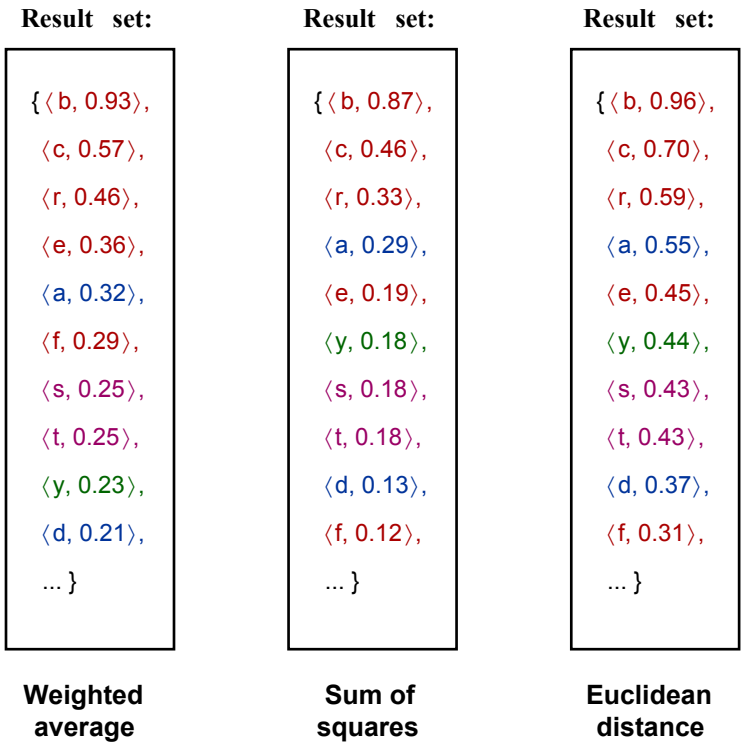


Fig. 2.18: Result sets for the summative union in Figure 2.17 using three different summative resolution functions. The linear weighted average produces a significantly different order than the others, which use quadratics. The last two produce the same order, but different drop-off functions.

2.6 Text and structured document searchers

Summative unions have been important in MARIAN collections in two key areas: text and structured document classes. Both of these are common in digital library applications. Text searchers make the vital connection between query strings entered by users and the multitudinous text fields throughout a collection. Structured document searchers make it possible to assemble matches on several areas or aspects of a single document into a single figure of merit for the document as a whole.

The match function of a text searcher assigns a weight to texts within a class based on how closely they match a query. In some search systems, where a query is a statement in some calculus, the strength of the match is determined by how well the text *satisfies* the query. The standard MARIAN text searcher is an example of a *free text searcher,* where a query is simply another piece of text and the strength of match is determined by how closely a text in the class resembles the query string.

The standard MARIAN text searcher is typical of free text searchers in that it ignores word order and sentence structure and assigns match weights based solely on the frequency of words and other terms within the two texts.[3] We will examine two variants of the text searcher, both based on a summative union searcher and both depending on situating text objects in a context of *OccursIn* links to terms.

In the case of text, we rely on parsers to generate *OccursIn* links for all texts in a class. This can be profitably thought of as a text class method, although in operational terms it is handled by "batch" programs in the collection building suite. However we view it, the operation of adding new text objects to a class like *SubjectEntry* or *Title* includes parsing the text and generating new *OccursIn* links to recognized terms. Once the class of *OccursIn* links in built, it can support both search methods.

Only a single parsing method is required for collection texts. The two text search methods differ in their treatment of the query text. In the first, simpler method, query text is parsed to a *bag* (multiset) of terms. The bag is then reduced to a weighted set where each element is a term and each weight is some function of the frequency of occurrence of the term in the query text. Each term in the query is then passed to the class manager for *OccursIn* links, which uses node activation to produce a weighted set of texts in the appropriate class linked to that term. These sets are scaled by the weights in the query set, and the results combined using a summative union operation. The resolution function assures that texts that have more terms in common with the query will, all other things being equal, receive higher weight, while also allowing individual terms of high weight either in the query or in the collection to dominate terms of lower weight.

The second search method differs in that the query is reduced to a weighted set of weighted sets of terms, where each inner set corresponds to possible interpretations for some substring of the query. This representation respects the reality that words in natural language are ambiguous, and that our best parsers still only guess at how a string of letters, numbers, punctuation and white space can be resolved into a sequence of words. Each inner set is thus a set of alternative guesses as to what the author of the query string intended: their weights within the set are determined by the confidence of the parsing algorithm in their chance of being correct. The outer weights represent the importance of the substrings in the query text. Thus the outer weights correspond to weights in the first method, while the inner weights trim the alternative interpretations by their relative likeliness.

The text searcher passes each inner set to the *OccursIn* class manager just as it passed a single term in the first method. The *OccursIn* manager uses weighted set activation to return a weighted set of texts matching *any* interpretation in the set, the result weights being determined by the strength of the *OccursIn* link combined with the confidence of the interpretation. The text searcher then performs the exact same summative union on these link sets as in the first method, developing a weighted set of documents that are good matches to likely interpretation. The final match weight combines how much of the query is covered by the match with how closely document terms match interpretations of query terms and how likely those interpretations are.

---

[3] The "standard" text searcher is only that: the text searcher used most frequently in MARIAN. The MARIAN representational schemes and searcher profiles can support other free-text searchers, as well as those that use artificial query calculi and those that respect more of the structure of language in the text. However, this particular free-text searcher has served us well, so it is the one we will examine most closely.

Both text searchers thus depend on a summative union searcher. The structured document searcher, by contrast, makes use of both summative union and maximizing union. Before we explain why, let us first consider the simpler case of an unstructured document.

A document is a sort of information object, so it is natural in MARIAN to represent a collection of documents with the same structure as a node class. Most documents exist in a context of links to other node classes: authors, references, originating institutions, or whatever structure the digital library designers have chosen to materialize. As such, a general document class searcher must handle search within context. This requires combining the results of local searches within the document class (most commonly on the text of the documents themselves) with searches on the pattern of links and nodes surrounding the documents.

2.7 Searcher internals revisited: set image tables and frontier managers

Sequencers and tables continue to be important in the design of summative union searchers, although in different roles. The sequencers in summative searchers are the same as described earlier, although we will introduce the related *frontier managers* at the end of this section. We will begin with tables, though, because summative union searchers require more complicated forms of tables than maximizing searchers.

Figure 2.19 introduces the basic table structure used in these searchers: a threaded table with multiple threads, one for each incoming set. As each set is developed, elements are attached to the end of the thread associated with that set. The set can then be reconstructed in its original order by following the thread from the beginning.

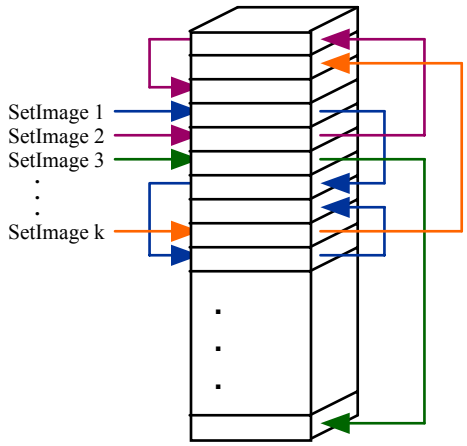


Fig. 2.19 In a set image table, each incoming set is associated with a thread that captures the order of elements presented.

The sets combined by a summative union searcher are assumed to be overlapping, since if they were disjoint no resolution function would be required. Thus it is possible for collisions to occur in the table. When this occurs in a SetImageTable, the elements are removed from the set image threads and placed into a separate thread for multiply hit objects (Fig. 2.20). Removing an

element from its original set image does not change its location in the table, since its location is determined only by the table hash function and the element's identity. It does, however, change its position in the weave of threads.

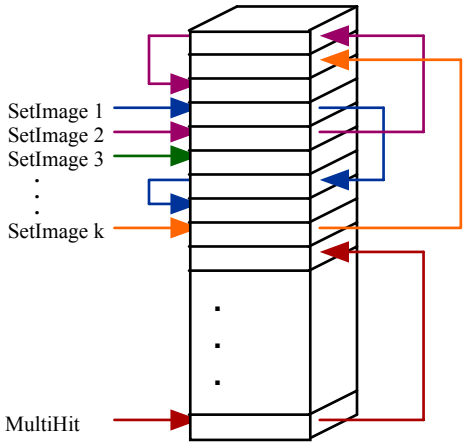


Fig. 2.20: When incoming sets are not disjoint, some accommodation must be made for objects in more than one set. In a SetImageAccumulatorTable, objects "hit" more than once are moved to a MultiHit thread and their weights summed.

A collision in the table means that the same element occurs in several sets. It cannot, however, occur on several set images, since we require the images to be disjoint. Thus we must adjust the references in the table cells – the links that make up the set image – to reflect the new situation (Fig. 2.21). First, the element is cut out of its original set image thread by setting the reference in the table cell before the multiply hit cell to reference the cell after. (This is implemented by adding "prev" pointers to each table cell, so that each thread becomes a doubly-linked list.) Then the element is added to the MultiHit thread. Since elements are unique within each incoming set, this action either removes it from some other set's image or moves it within MultiHit. As elements common to more than one incoming set are moved to the MultiHit thread, the set image threads become imperfect copies of the original sets. It is important to note, however, that they are still in weight order: removing elements does not change the non-increasing nature of the sequence.

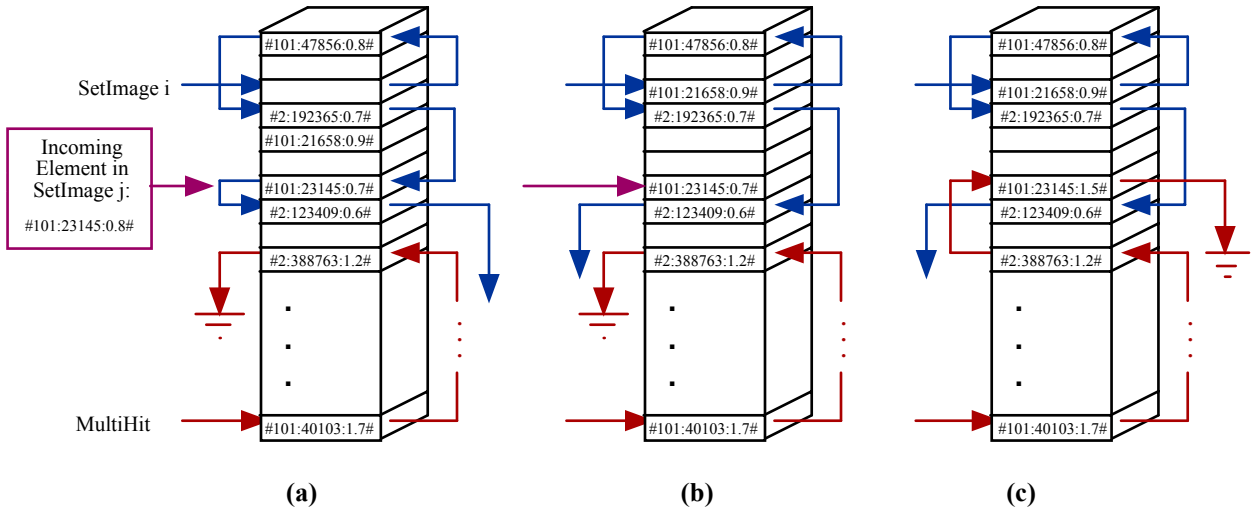


Fig. 2.21: When non-disjoint sets are added to a set image table, collision may occur. An example collision is shown in (a), where element #101:23145# is found in set j with a weight of 0.8. But #101:23145# is already in the table (with weight 0.7) as an element of set image i. Thus, in (b), the element is cut out of set image i. In (c) the new weight is summed with the existing and the element hooked onto the end of MultiHit thread.

The cells in SetImageTables, like the InsertionOrderedWtdObjTable described above, contain a weight. In the singly-hit cells of the set image threads, the weight is the same as the incoming weight. When cells are multiply hit, however, we must choose what to do with the multiple incoming weights. For summative union searchers, the operation is simple: when an incoming object is discovered to be already in the table, the incoming weight is added to the existing weight. This is true whether the element is on a set image thread or the MultiHit thread. Thus the weight component of each table cell functions as an *accumulator,* and the table is called a SetImageAccumulatorTable.[4]

Two questions remain about MultiHit: should it be sorted or kept unordered, and should new collisions on MultiHit be treated differently than new collisions on a set image thread? We take the second question first. A new collision on the MultiHit thread will not require the element to change threads, but it will change the weight value of the element, as the new incoming weight is accumulated. If MultiHit is maintained in a sorted order, most collisions will thus create violations in the order. Removing the element from MultiHit and reinserting it using the same algorithm as insertion of a new element is thus a reasonable procedure. Such a procedure seems unnecessary if MultiHit is not ordered, however. On the other hand, if MultiHit is unordered, then the cost of this unnecessary operation is small. Removing an element from some arbitrary place on the MultiHit thread and reattaching it to the end of the thread is a cheap **O**(1) operation involving redirecting a handful of references. Conversely, simply establishing whether or not an element is on the MultiHit thread requires at least one comparison operation, *and needs to be performed for every collision* not just for collisions on MultiHit. Furthermore, any efficient scheme for establishing whether or not an element is on MultiHit requires an expenditure of

[4] Simply adding the new weight to the old introduces the possibility that the result may be greater than topWt. Thus the accumulator is technically not an object of class *Weight* but of *ExtendedWeight*. Any properly implemented summative union searcher will address this as it removes elements from the table by applying some appropriate post-function to reduce the accumulated weights to legal values.

storage of at least one bit per table cell. Thus in all cases, it is reasonable to treat MultiHit collisions the same as new collisions.

Whether MultiHit should be ordered or not depends on the particular use to which the accumulator table is put. If the table is to be filled before any elements are taken from it, sorting MultiHit as it is built causes extra work every time a MultiHit collision happens. In this case, it is most reasonable to wait until all elements have their final accumulated weight before sorting MultiHit. If on the other hand elements will be removed from the table during the filling process, then an order will need to be established for MultiHit whenever elements are called for. We will discuss the resulting structure during the exposition of opportunistic searching in the next section.

One final component to be introduced is the *frontier manager*. We have seen frontier management already in the control for the set mapping sequencer, where the key and map sets were developed only to certain related minimum weights. A frontier manager consists of a similar control unit with no priority queue. It functions as a gatekeeper for a collection of incoming set presentations, letting through only elements with weights above a certain threshold value. Frontier managers can work with a set collection or a set mapping situation: in either case their sole function is to maintain a frontier through the incoming sets and to extend that frontier when presented with a new threshold value. When a frontier manager created, the first threshold is taken to be at topWt; thereafter each threshold is expected to be at a lower value. This functionality is different from sequencers, which always return exactly one element when called. Depending on the distance between the new and last threshold and on the drop-off functions of the incoming sets, a frontier manager may return anywhere from zero elements to the entire contents of all the incoming sets. Frontier managers are not required to return elements in any order, so no priority queue is needed. Instead, a frontier manager simply works through each set in its input collection, returning all elements above the new threshold value (and, in a set mapping situation, adding new sets to the mix as needed).

2.8 Anatomy of summative union searchers

As with maximizing searchers, summative searcher efficiency depends upon additional information about the incoming sets. In the case where no information is available, the problem described above dominates: it is possible for an element that occurs at the very end of the incoming sets to receive a final weight among the highest in the set if that element only occurs in a sufficient number of incoming sets. This informs the default strategy for summative union search: first run all incoming elements from every set into the searcher table, and only then get sophisticated.

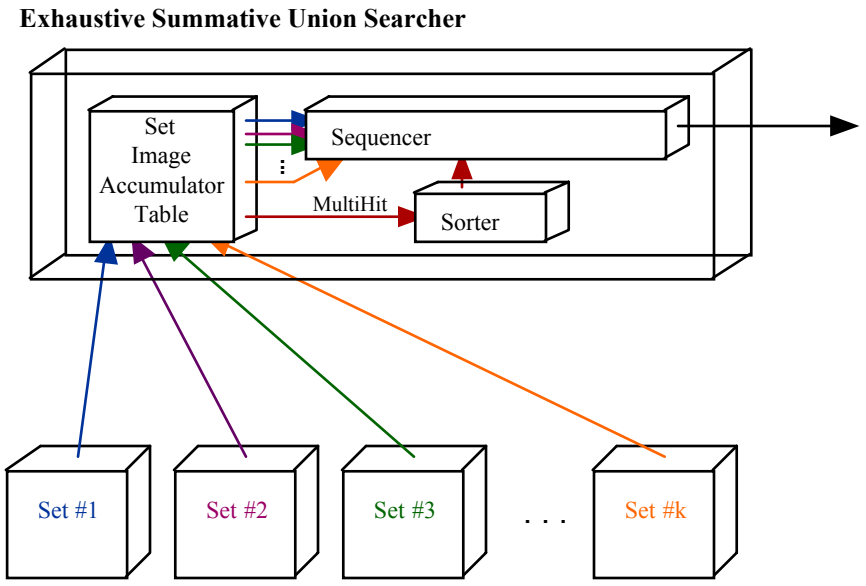


Fig. 2.22: An exhaustive summative union searcher uses tables and sequencers in the opposite configuration from a maximizing union searcher. Incoming sets are run into a set image table, with weights accumulating in the MultiHit thread. MultiHit is then sorted, and the results merged lazily with the remaining set images by a set collection sequencer.

The exhaustive summative union searcher (Fig. 2.22) follows this strategy. As each incoming set is added, it is completely developed into a set image in a SetImageAccumulatorTable. When an element is discovered already in the table it is unhooked from its current thread and hooked onto the end of the MultiHit thread. At the end of the k^{th} set addition, the table consists of k set images, each of which includes only elements that occur within that set and no other and each of which is in non-increasing weight order, together with the MultiHit thread in no particular order. All k+1 threads are disjoint, since any common elements among the incoming sets have been moved to MultiHit, where they occur exactly once. Thus a standard sequencer can merge them efficiently and lazily into the final union set, once MultiHit has been put into weight order.

The task of ordering MultiHit is the second biggest consumer of resources in the operation of the searcher. (The first is running all the elements into the table, which requires $\boldsymbol{O}(\sum_i N_i)$ operations within the searcher, plus any costs involved in developing the incoming sets.) Ordering MultiHit completely requires at least $\boldsymbol{O}(N_{MultiHit} \cdot lg\, N_{MultiHit})$ since MultiHit has no internal structure that would allow us to improve on general sorting performance. It can be argued, though, that $N_{MultiHit} \ll \sum_i N_i$, so even completely ordering the thread is a secondary cost. And if the ordering cost can be distributed over elements returned, the expected cost of returning a single element becomes $\boldsymbol{O}((k+1) + lg(N_{MultiHit}))$ for a linear search sequencer and $\boldsymbol{O}(lg(k+1) + lg(N_{MultiHit}))$ for a heap sequencer.

(In fact, this rough estimate is too simplistic in several ways. First, the $lg(N_{MultiHit})$ term only applies to elements that are found on MultiHit, of which there are only a few relative to the size

of the union set. On the other hand, since the elements in MultiHit have accumulated weights, most of MultiHit will be returned early in the development of the union set, so that cost is "front-loaded." Finally, as long as high-weight elements are coming off MultiHit, the sequencer will not be doing any work and so the $(k+1)$ term will not apply. A more exact model would proceed from these observations to estimate that the development of the entire set is $O(N_{MultiHit} \cdot lg(N_{MultiHit}) + (N_{Union} - N_{MultiHit}) \cdot k)$ and then amortize the cost over union set development. However, we can do fairly well by noting that both k and $N_{MultiHit}$ are enough smaller than N_{Union} that they can be conflated, and we will not be far off in saying that once the table has been filled the cost of developing elements is a small constant compared to the size of the union set.)

Producing elements efficiently from a filled table is cold comfort, however, when filling the table is an expensive operation. And filling the table is indeed expensive, both because it requires $O(\sum_i N_i)$ table operations and because it involves completely developing the incoming sets. If summative union is the last operation performed by the search system, all the advantages of lazy evaluation of sets up to that point will be for naught. Fortunately, this is not generally the case. A use study in 1996, during a time when MARIAN was being widely used as a pubic library catalog for Virginia Tech, found that while users often performed structured-document searches, they were more likely to use cross-component searches (requiring maximizing union as the last operation) then multiple-component searches (requiring summative union). As mentioned above, all text searches are also summative but they take place on smaller incoming sets, and in general on node-activation sets that can be retrieved directly from link class databases. And in any case, when elements at the end of incoming sets are likely to combine to produce weights at the top of the union set, the only alternative to discovering those elements is to risk failing to report good matches to the user.

Suppose, however, that elements deep in the incoming sets were unlikely to produce any changes to the top portion of the union set. Then it would be possible to run only some of the incoming elements into the union searcher, deferring full development of the incoming sets until later. That is the strategy behind the MARIAN opportunistic summative union searcher (Fig. 2.23). For now we say only that the abovementioned situation occurs less often than one might hope, and that it happens only when there many incoming sets. Users are unlikely themselves to structure searches that require summative unions of large numbers sets, but there are several automatic techniques to improve retrieval, including query expansion and relevance feedback, that do produce this situation. What is more, the situation can be recognized at the time of searcher construction. Thus, like choosing which priority queue to use in building a sequencer, this choice can be made automatically and at run time, so the alternatives impose no burden on system administrators or users. And as it turns out, the searcher can be constructed with only a small complication to the SetImageAccumulatorTable and the addition of a frontier manager.

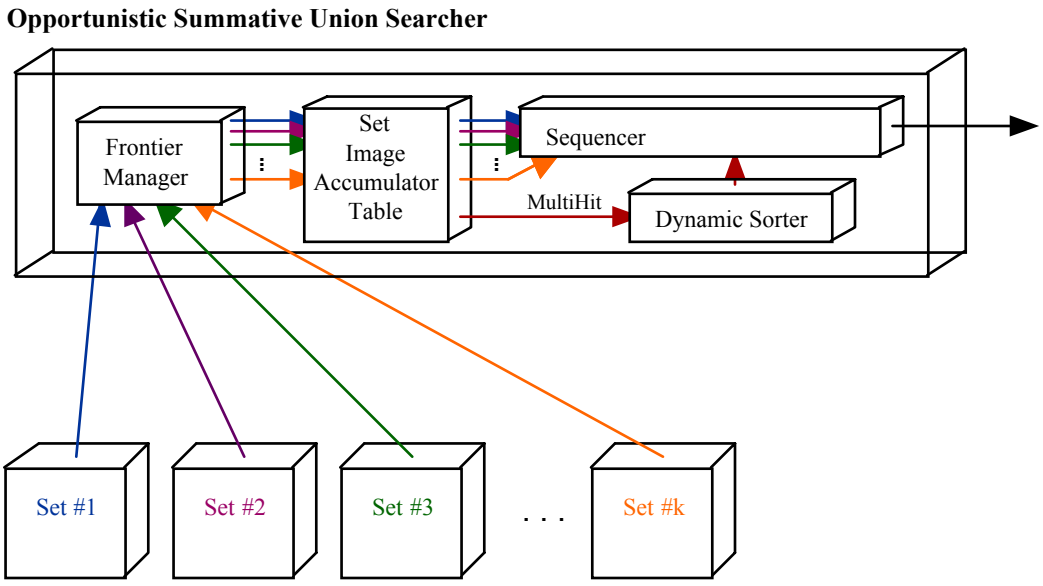


Fig. 2.23: The opportunistic searcher uses information about expected set overlap to draw a frontier through the incoming sets. The accumulator at any point holds only a fraction of all possible elements in the union, and MultiHit must be dynamically sorted as needed.

Unlike the exhaustive summative union searcher, the opportunistic searcher can make effective use of set mapping (Fig. 2.24).

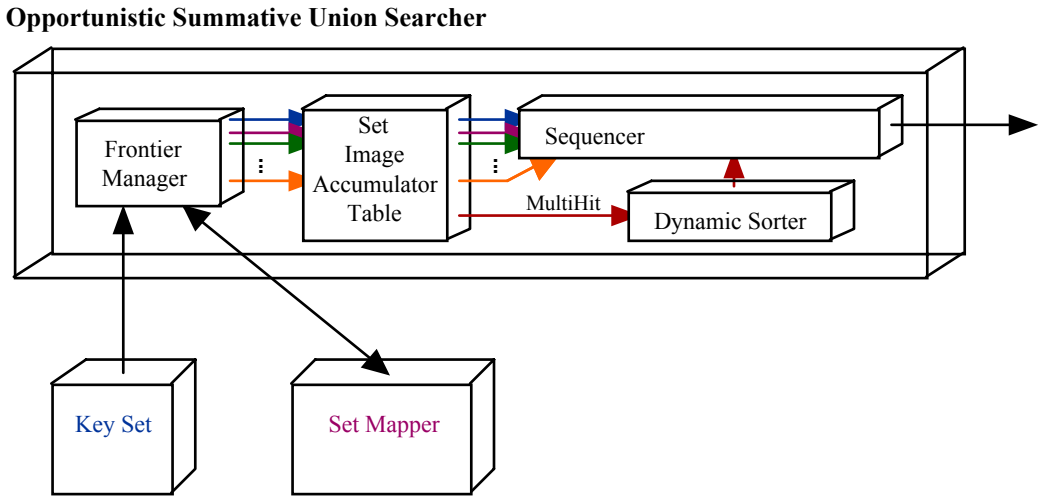


Fig. 2.24: Opportunistic searchers can also work effectively with mapped sets.

Neither the exhaustive nor the opportunistic searchers is quite ready for prime time as described. For one thing, they suffer the same lack as the DisjointUnionSearcher: the stream of returned elements is produced by a Sequencer and thus is not easily shared among clients. The resolution of this problem is the same as for the DisjointUnionSearcher: cache the elements on their way out and share the cache. In addition, the searchers as presented are not *summative* union searchers, but *summing* union searchers: they do not implement the general class of summative

functions $g\left(\sum_i f(\omega_i)\right)$, but only simple arithmetic sum $\sum_i \omega_i$. To generalize them we need to add the pre-functions $f()$ to the inputs and the post-function $g()$ to the output (Fig. 2.25). For average, weighted average, and relative entropy resolution functions all these computations are scalar multiplication; for vector cosine, the pre-functions are scalers and the post-function a square root followed by scalar division.

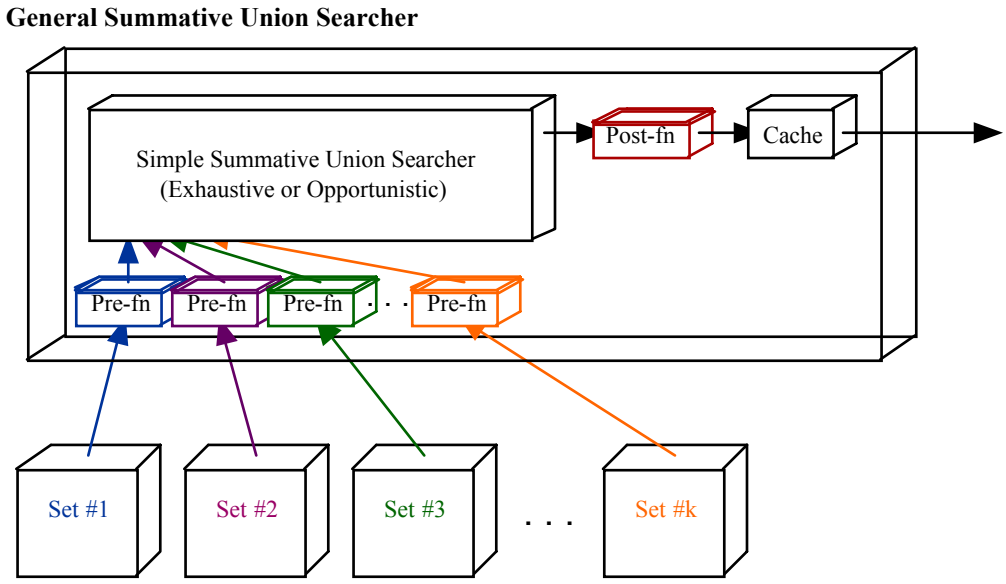


Fig. 2.25: In the general case, summative union requires applying some pre-function to each input and a post-function to the output. A cache on the output allows presentations to be generated for multiple clients.

One nagging question remains: are we *ever* justified in failing to return a high-ranking match to the user? Especially in a digital library application, users expect relevant items to be returned to them at or near the top of a match set. An opportunistic searcher may violate that expectation: there is a non-zero probability that items belonging at the top of the match set will not be discovered until the bottom. As creators of digital library systems, are we ever justified in using such an algorithm?

One possible answer is that we are justified in using opportunistic search when the uncertainty it adds is not significant in light of existing uncertainty. Existing uncertainty is relatively high in all information retrieval systems, if only because the underlying data are so sparse and because small changes in term frequency can cause large variations in term-text association weights. Further, the numbers and functions we use to represent strengths and combinations of evidence are only superficially related to the intensional processes a user goes through in judging relevance. All in all, no search system is ever completely certain to retrieve all relevant results and no ranking system is guaranteed to correctly order results. Given these confounding factors, we may consider ourselves justified in further relaxing our system behavior in order to increase efficiency. Whether we actually choose that alternative in any particular digital library system will depend on how willing we are to make that exchange.

Chapter 3: Indexing Large Collections of Small Text Records for Ranked Retrieval

3.1 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.

3.2 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. 3.1). 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 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 judgment that the title was a more important part of the description of a work than any notes it might have.]]

3.3 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 $O(n \cdot \log(k))$ 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.

3.4 Data Processing and Characteristics

There are three major steps in preparing MARC records for retrieval (Fig. 3.2). 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 3.3 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 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. 3.4). 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 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. 3.5). This confirms our confidence in the effectiveness of LIMA.

Figure 3.6 shows how terms are assigned to classes. Figure 3.7 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 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 3.8, 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. 3.9). 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 3.10 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 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. 3.9). 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 3.9 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. 3.11).

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 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 Ð 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.

3.5 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. 3.12), 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.

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. 3.13) 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.

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 Kučera, we will only believe that the matching formula is well-behaved if it matches "Francis" against Francis, and "Kucera" against Kučera; 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:

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

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:

$$(<a_i>, <b_i>) = \sum (IDF(i) \cdot a_i \cdot b_i)$$

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:

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

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.

3.6 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 3.14 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.

3.7 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.

3.8 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.

3.9 References

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 International 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.

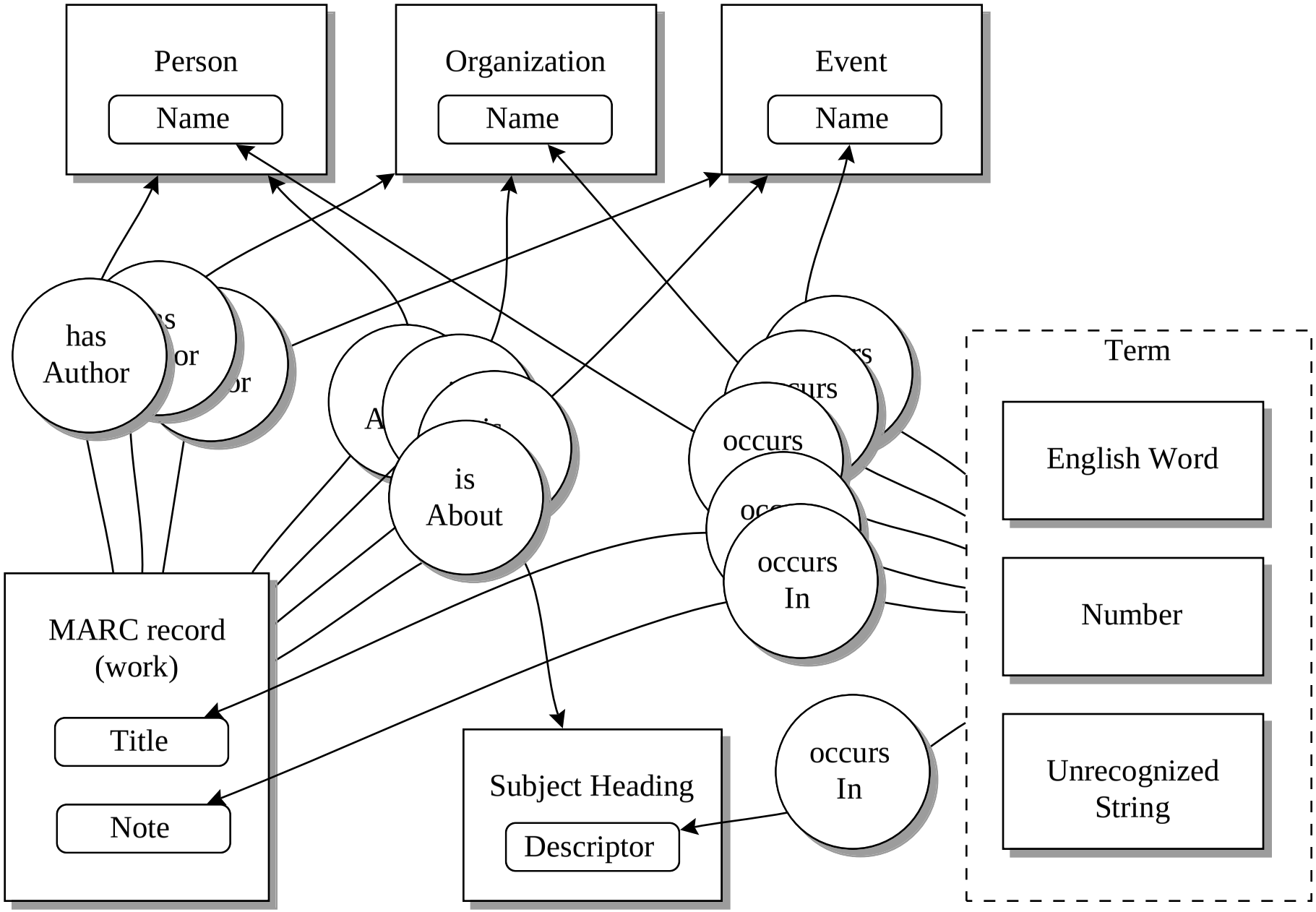


Fig. 3.1: Entities and links in MARIAN. Composite entities are shown as rectangles, links as lines with circles, and text fields as lozenge shapes.

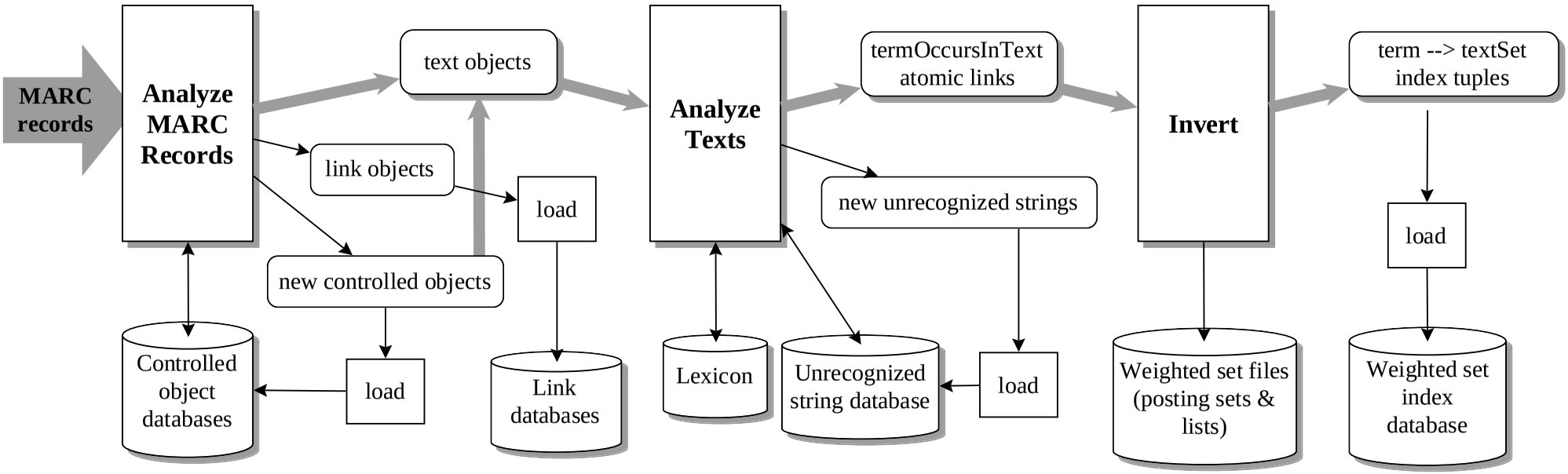


Fig. 3.2: Major processing steps, intermediate files, and class databases for MARIAN.

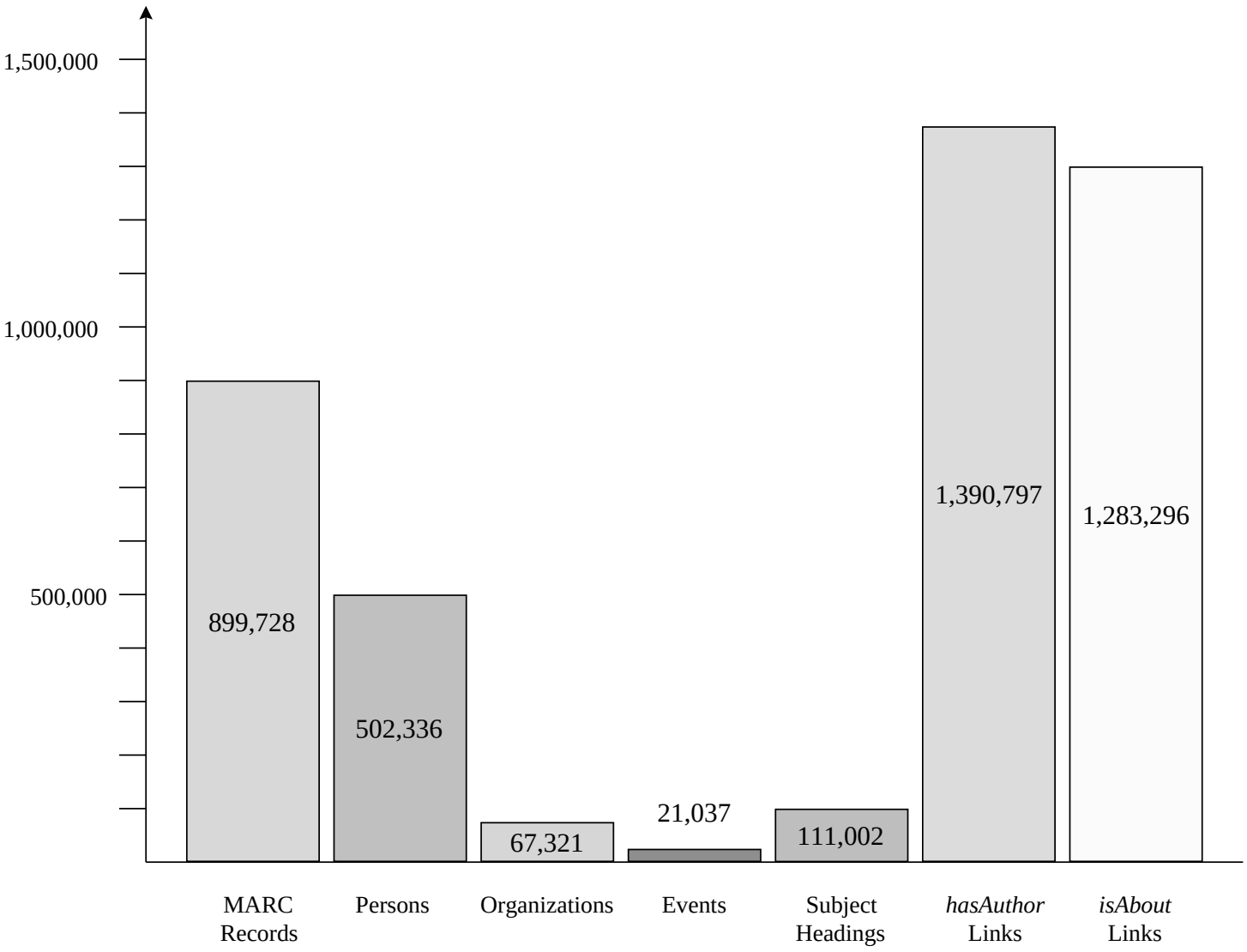
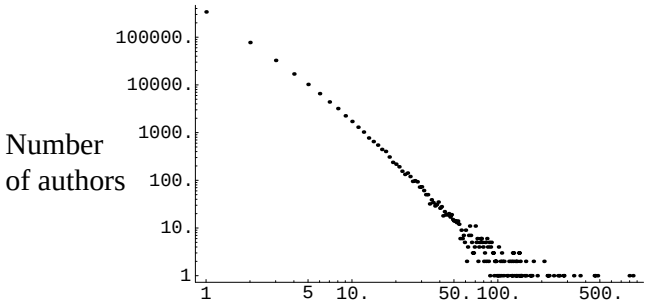
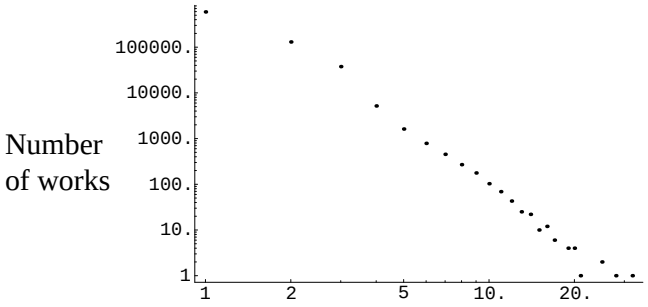


Fig. 3.3: Number of objects in each of the classes produced by MARC analysis.

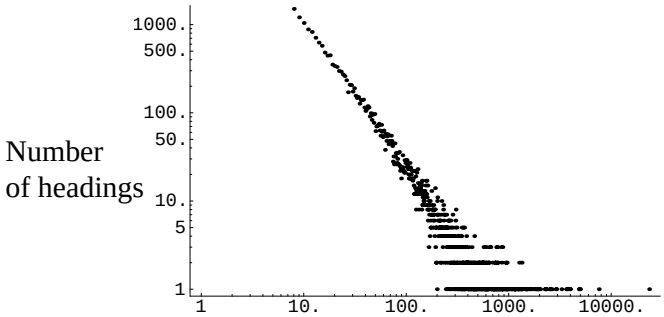
Number of Works per Personal Author



Number of Personal Authors per Work



Number of Works per Subject Heading



Number of Subject Headings per Work

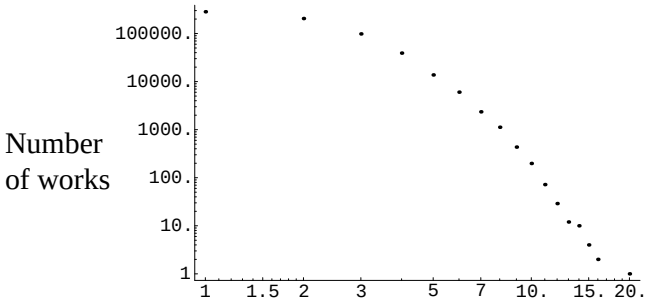


Fig 3.4: Distribution of links for *isAbout* and *hasPersonalAuthor* classes. The Zipf-like distributions show up clearly in these log-log plots for all except the number of subject headings added to a cataloged work. The shape of the curve there is a result of controlled cataloging practices, which encourage an "optimum" two to three headings per work.

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. 3.5: 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 3.6: 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.

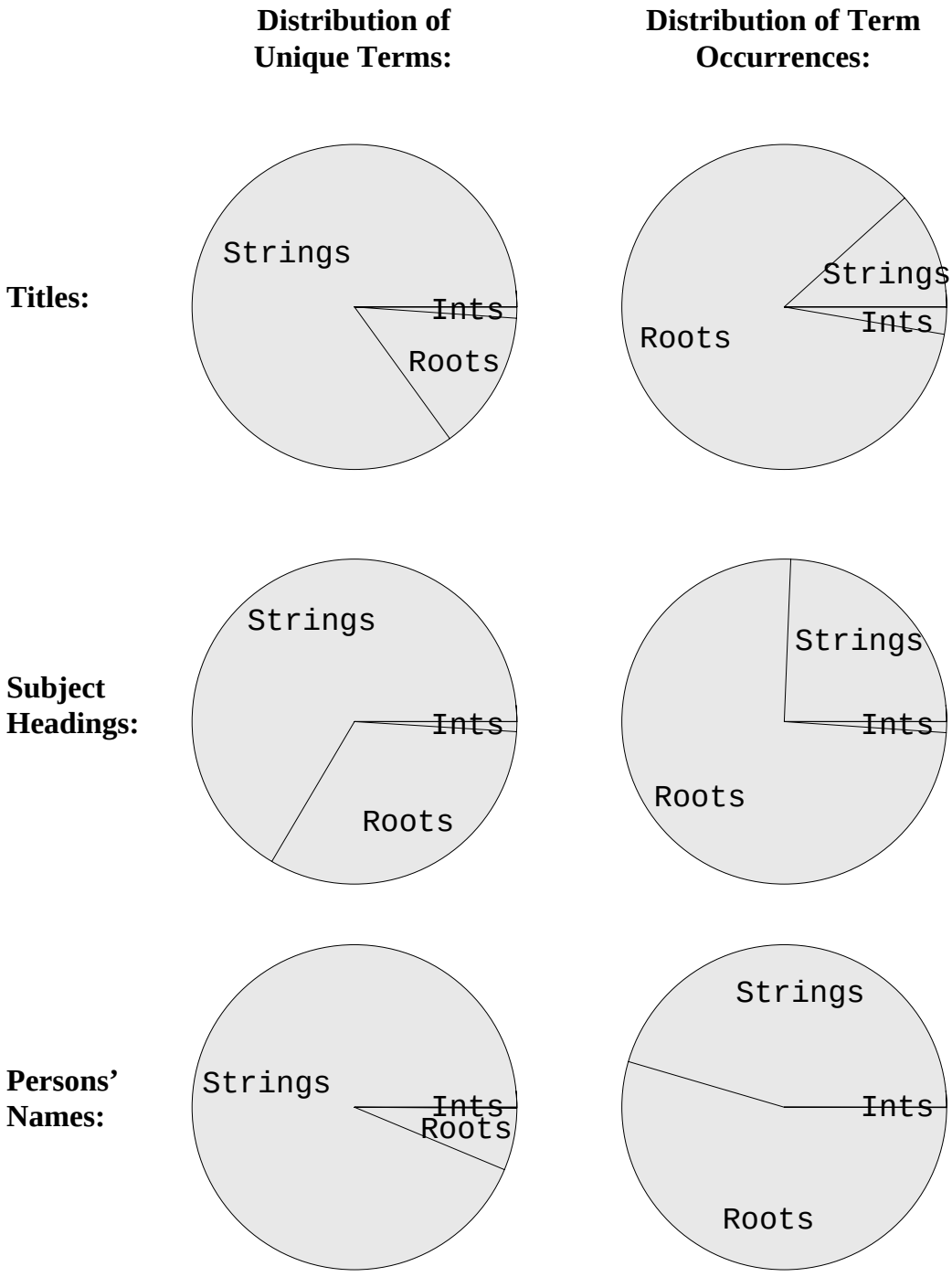


Fig 3.7: Distribution of terms by type for three categories of data, showing relative numbers of integers, English roots, and unrecognized strings. Other categories were too small to be significant.

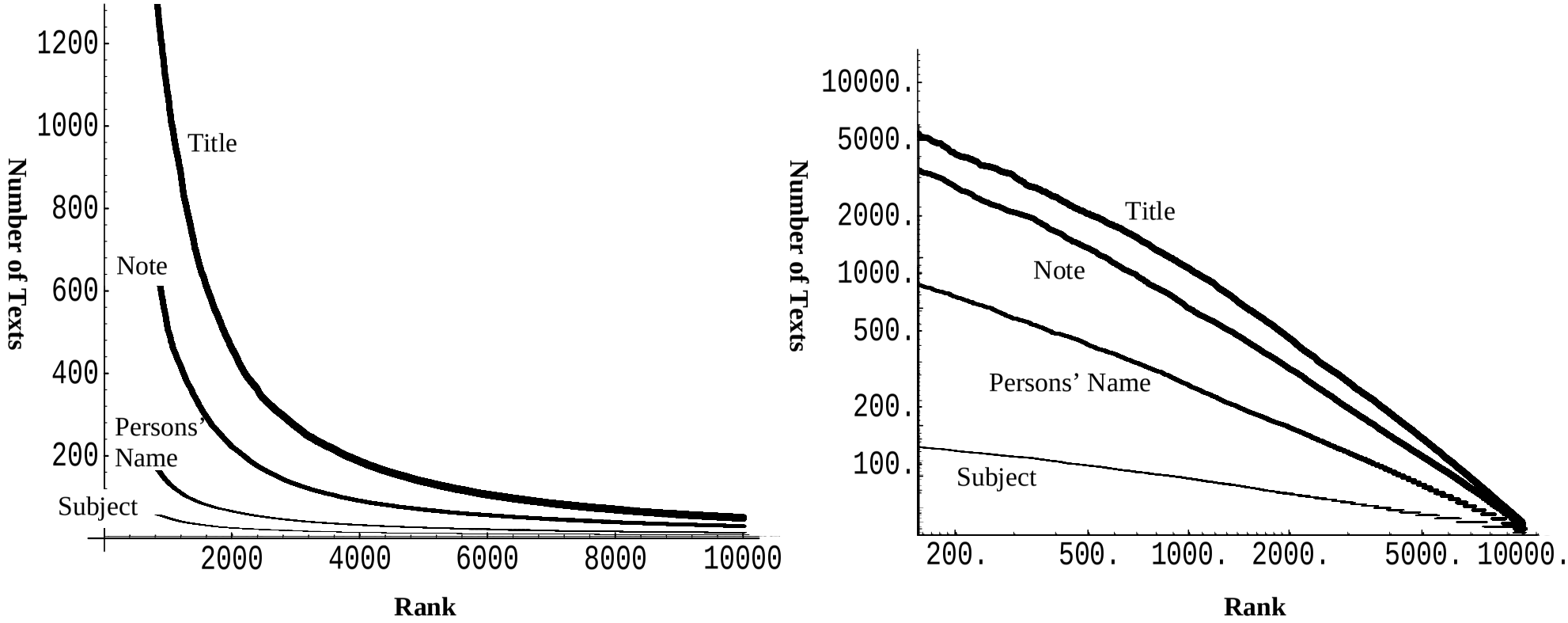


Fig. 3.8: Observed rank/frequency curves for text in the title and note fields of Virginia Tech MARC records, the names of unique persons, and for unique subject descriptors. The left-hand graph, with linear axes, shows the characteristic shape for the central portion of the data set. When the entire 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.

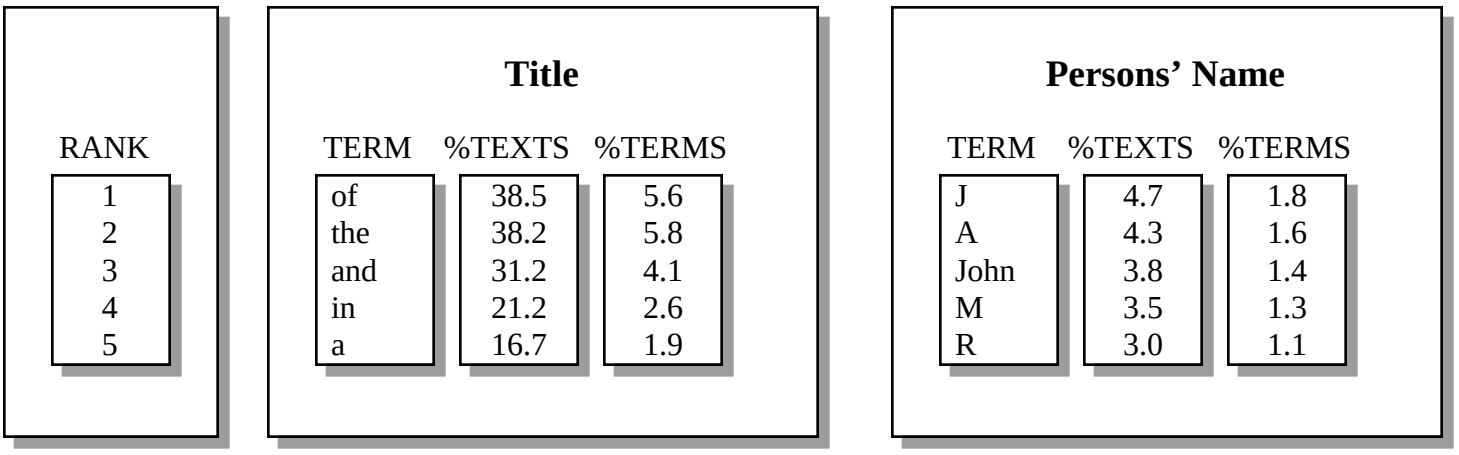


Fig. 3.9: Top-ranked five terms in *MARC Title* and *Persons' Name* text collections, together with the percentage of texts containing the term and the percentage of all the term instances in the collection that it accounts for.

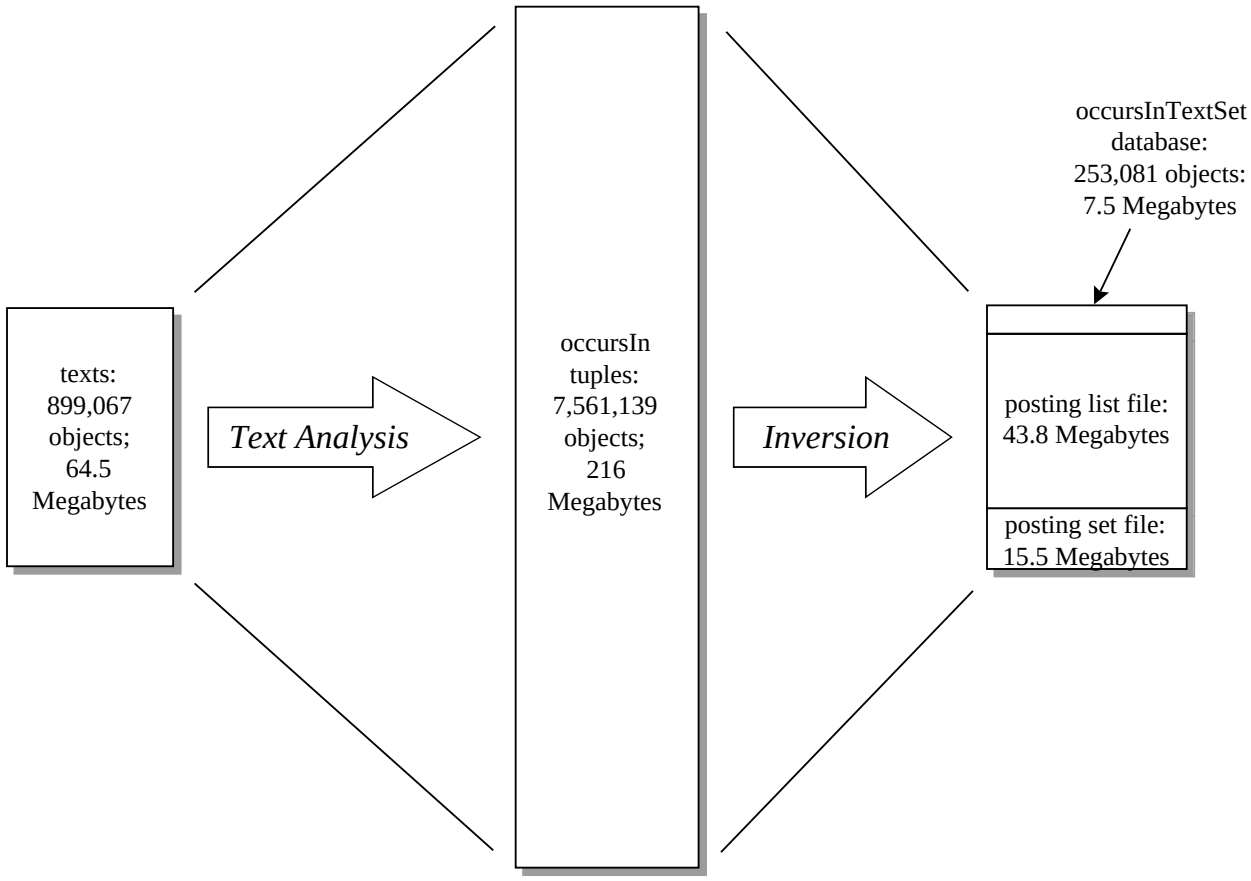


Fig. 3.10: Expansion and contraction of data during text indexing. Vertical size of boxes is proportional to size in Megabytes. Figures are for title collection, but are typical.

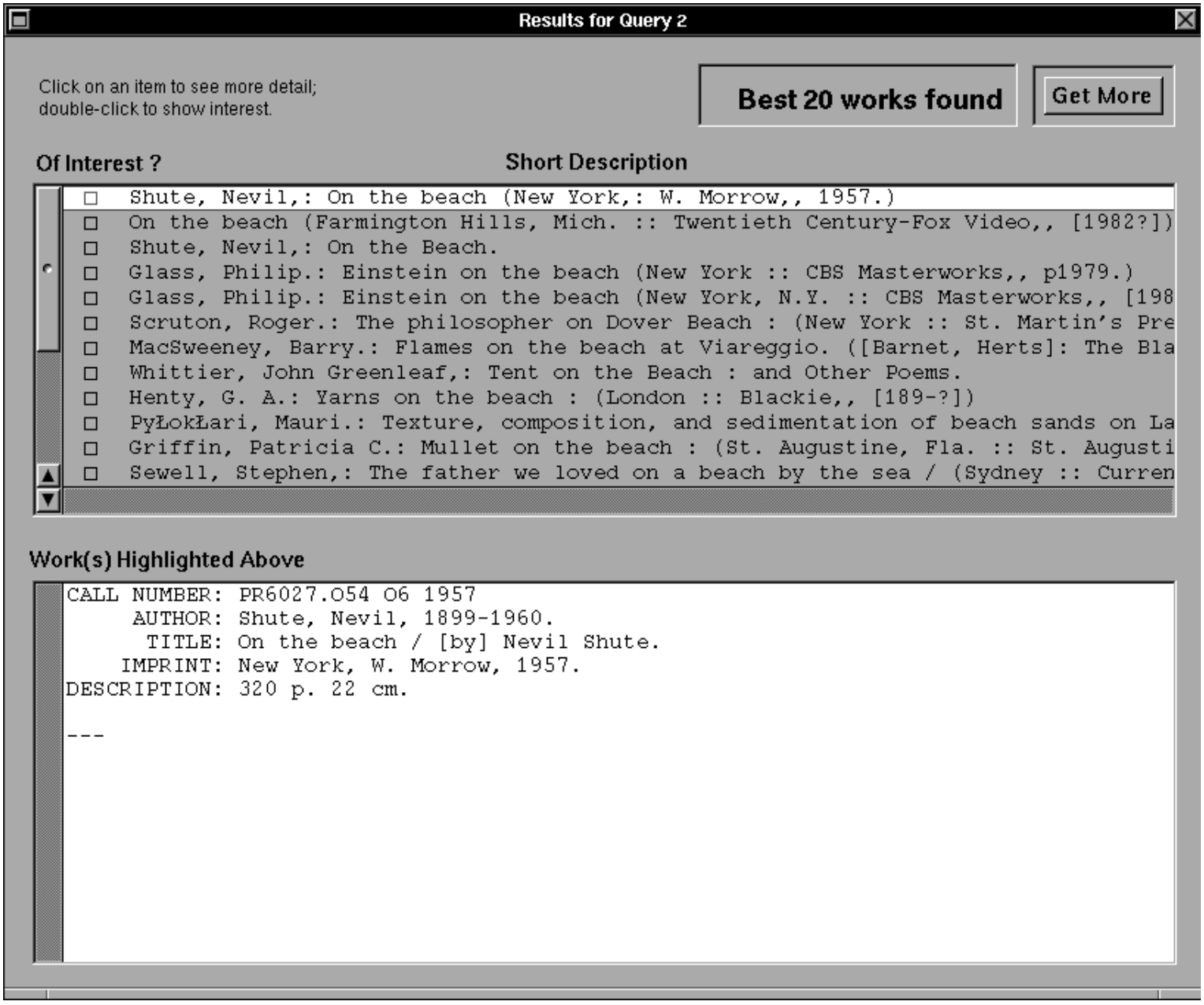


Fig. 3.11: Results from MARIAN for the query "**Words in Title:** On the beach."

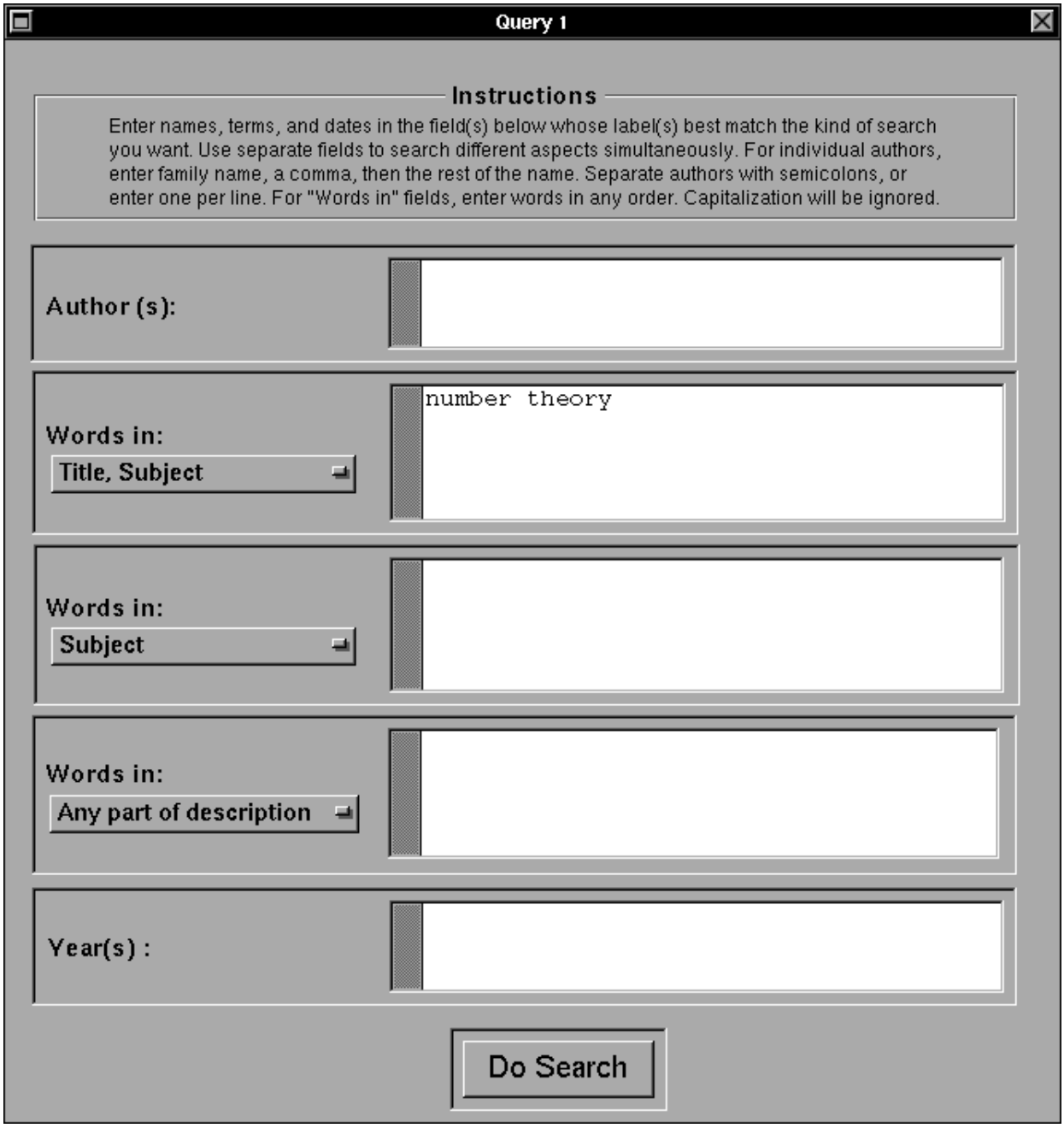


Fig. 3.12: A multi-part MARIAN query. Equivalent queries could be constructed using this form in a number of ways, most simply by setting the coverage of the text field to "Title" and copying the text to the field below.

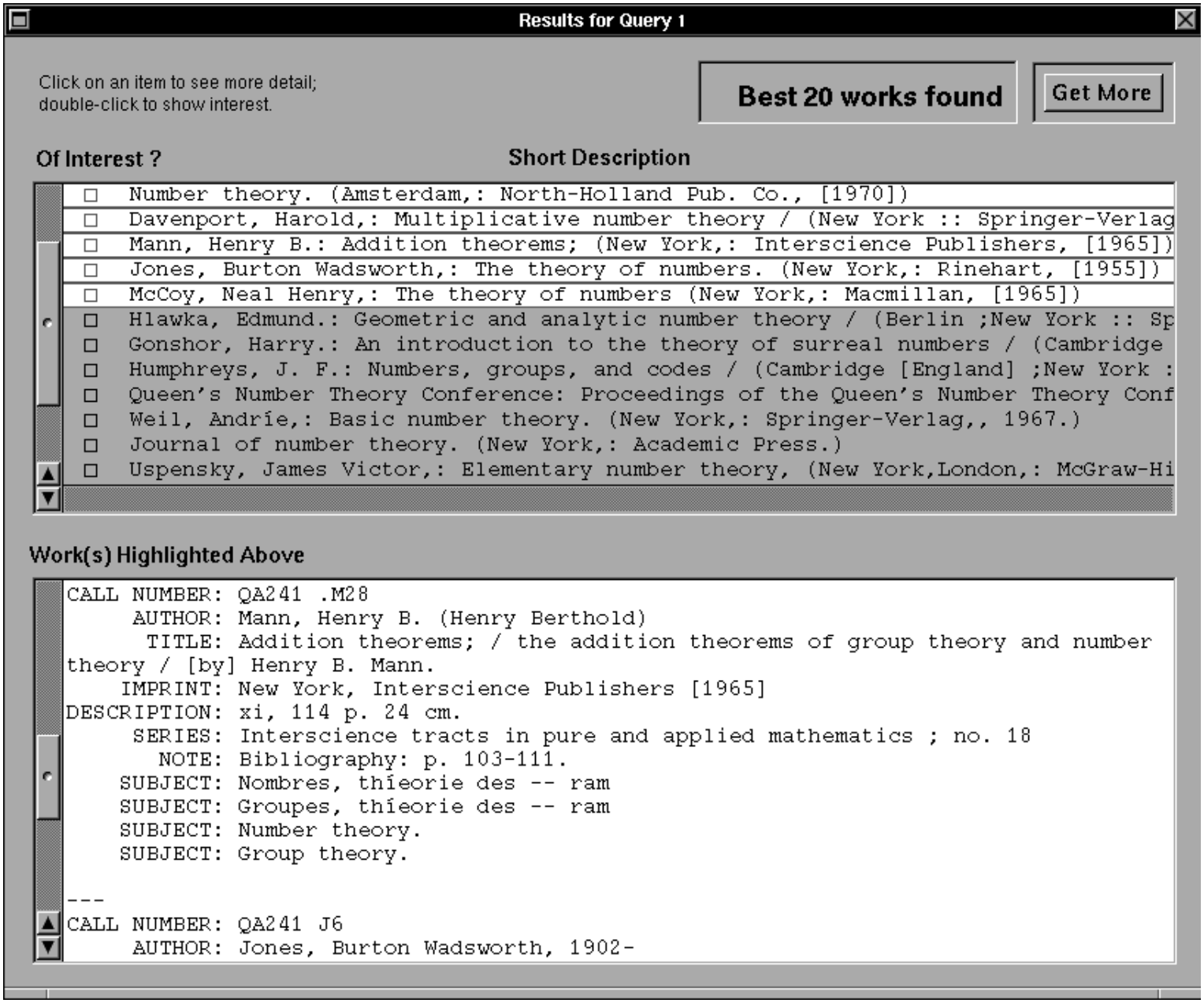


Fig. 3.13: Results of the query in Figure 11.

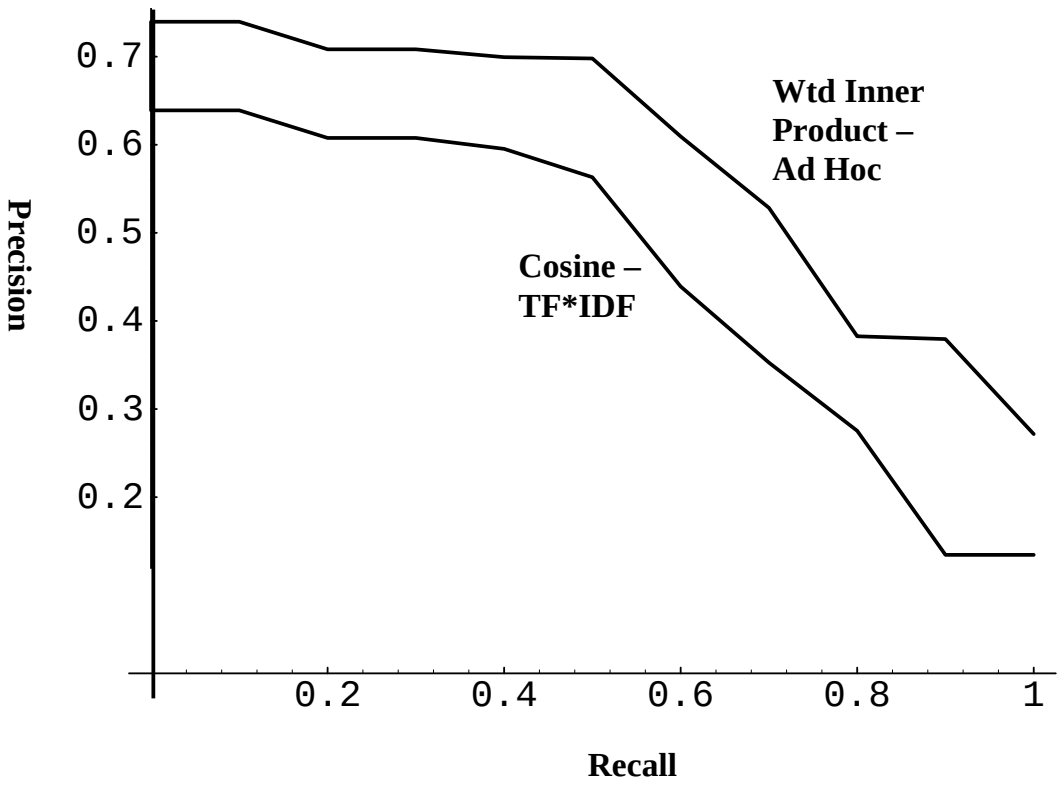


Fig. 3.14: 11-point recall-precision curves for MARIAN running two different similarity functions: vector cosine with TF*IDF weights and weighted inner product with ad hoc information-theoretic weights.

Chapter 4: Why Stopping Rules Don't Stop

Ranked retrieval systems, particularly those based on approximate matching functions, have many attractive features. They are easy to use, as they can often be interrogated effectively with simple lists of terms. They are tolerant of error in queries, and they do not require users to attend to word frequencies in order to put together queries. And they provide a good interface to large document collections, where even the most carefully thought out Boolean queries may retrieve thousand of documents. Ranked retrieval systems also may retrieve thousands of documents (even tens of thousands) from the same query, but they can present the user with the top few documents and thus allow him or her to continue.

The key to this behavior involves use of a heuristic weighting function that ranks documents in the collection by their similarity or relevance to the user's query. A number of these functions have been proposed over the years [SALT&BUCK88]. Yet none have been shown to be always better. Different types of queries, different types of document collections, and even different statistical conditions of the documents in the collection all have an impact on which weighting function works best in a situation. A common feature of the functions proposed, however, is that they are typically summative in nature. The overall figure of merit for

the relevance of a document to a query is expressed as a sum over the parts of the query, be those document attributes, clauses in an extended Boolean representation, or simply words in a text expression. This is a problem.

The most common types of ranked retrieval search engines use an inverted file structure. The components of a document are collected and indexed as *posting lists* of document-weight pairs retrievable by component type.* If the relevance of a document to a query is to be determined by a sum over all components in the query, then the exact rank of a document depends of its weights in all the posting lists. Computing exact weights for all matching documents requires touching every item in every posting list at least once. Many posting lists are quite long. In a collection of a million documents, the most common terms in the language can have posting lists with hundreds of thousands of elements. Computing exact weights can require many operations, and most importantly many disk accesses.

This problem can be attacked on at least two fronts: a search engine can sacrifice some precision in final weights, and thus in final ranks, or it can avoid calculating some fraction of the text weights. Both these strategies play into the method of *stopping rules* first proposed by Smeaton and van Rijsbergen [SM&vRIJS81] in the context of finding the nearest neighbors to a text document. In outline, the idea behind the method is to examine first those lists and postings that will have the most effect on the final weights of any document in the result set, and to apply some sort of heuristic rules periodically throughout the search in hopes of being able to stop before the low-weight lists or postings have been examined. The method was refined and applied to the problem of ranked retrieval by Buckley and Lewit [BUCK&LEW85], but with disappointing results. They found in experiments on standard collections that only “drastic” sacrifices in the dependability of final weights produced a notable improvement in performance. Further refinements of the method were made by Wong and Lee [WONG&LEE91, LEE&WONG91], who also found the performance disappointing. Harman and Candela [HARM&CAND90] used a related method to prune search results after all postings had been examined. Their improvements were greater, but still less than might have been expected.

***** For the sake of simplicity, we will restrict our discussion here to the problem of matching a single piece of query text to a single piece of document text. In this case, the document components are the terms that make up the text, and posting lists of <documentID, weight> pairs are associated with individual terms. The same considerations apply when document components include names, dates, structured texts, and so forth. Extended Boolean systems raise further problems, but the ones that we examine are still pertinent.

Harmon and Candela's experiments differed from both Buckley and Lewit's and Wong and Lee's in that they were made on a large collection of real-world documents. Encouraged by their results, and bolstered by our own further refinement in the stopping conditions, we decided to include an opportunistic search engine in the MARIAN system.* Unfortunately, our experience in MARIAN searches only confirmed the experiments of Buckley, Lewit, Wong and Lee. In most cases, searches either could not be stopped short of complete posting list exploration, or could only be stopped when the vast majority of the postings had been examined. The search engine could only be persuaded to stop significantly before the end of the search by setting the acceptable error in weights so low that an initial result segment of 20 documents was virtually guaranteed to miss at least one document that should have been there.

Such poor performance is all the more mystifying since stopping rules have been shown to perform well in their original context, that of clustering documents into nearest-neighbor groups [LUCARELLA88, RASMUSSEN92]. This paper attempts to explain why stopping rules work less well in the context of ranked retrieval. We first present a model for the inverted file structure of a text collection and the retrieval process, together with supporting evidence from the document collection used in MARIAN. We then introduce our opportunistic search method, together with our refined stopping rules. The final section is a statistical analysis of when the stopping rules can legitimately terminate a search before all postings have been examined.

4.1 Model

The individual posting lists in an inverted file system and the results of a ranked retrieval have the same formal structure. Both are examples of *weighted object sets:* sets of objects, each of which has an associated weight. In this paper we will take *weights* to be real numbers in the interval [0,1], where 0 means least interesting and 1 most interesting. As it happens, zero

---

***** MARIAN [FOXetal93] is a ranked retrieval library catalog system developed at the Virginia Tech Computing Center. Its purpose is to provide an alternative public catalog for easy network access where approximate matching methods serve as an alternative to more rigid access points. MARIAN is implemented as a set of clients communicating with a distributed server. The server includes both exhaustive and opportunistic search engines, as well as several other innovative features. The search results in this paper are drawn from our first full-size database: a 1994 snapshot of the full Virginia Tech library catalog of 960,000 MARC records. Records are indexed by title; personal, corporate, or conference author; subject; and notes. The posting lists shown are from the title index.

weights never occur in weighted object sets, but it is useful to have the value in the model. We purposely avoid defining objects. In actual implementations, set elements tend to be object surrogates such as ID numbers anyway; thus both the implementation and model make no commitment on objects beyond identity. We can envision weighted object sets in two (generally isomorphic) ways: either as sets of <object, weight> pairs; or as structures $\langle S, \omega \rangle$, where S is a classical set of objects and ω a *weighting function* that maps each object in S to a non-zero weight. The first picture reflects most common implementations of posting lists and result sets; the second is more amenable to formal manipulation. We will move back and forth between them at will.

Most of the structure of a weighted object set derives from the underlying set S. As in a classical set, the objects in a weighted object set are unique. Weighted object sets take part in the classical set operations of creation, test for an element, intersection and union. In the case of intersection and union, the result is an underlying set that is the classical intersection or union of the component underlying sets, together with a weighting function derived by some *weight combining function* from the weighting functions of the component sets. Any ranked retrieval searcher implements a weighted object set union defined by some particular weight combining function.

The most marked enhancement in a weighted object set is that the weighting function implies a (pre-)order in the elements. Elements with a higher weight can be said to rank above elements with a lower weight, and it is often useful to explore a set in this order. Thus each weighted object set is equipped with an iterator that returns elements in non-increasing weight order. This iterator is not necessarily unique. When a set includes more than one object with the same weight, different iterators may return those elements in any order.

4.1.1 Posting Lists as Weighted Object Sets

If we explore a weighted object set in iteration order, we will see a sequence of non-decreasing weights. We can regard this sequence as a discrete function over the set, considered as an ordered collection. We will refer to this as the *weight attenuation function.* In actual posting lists, the attenuation function is always a stairstep function (Figure 4.1). Each stair tread corresponds to a particular combination of document characteristics. In systems using TF or TF-IDF (Term Frequency – Inverse Document Frequency) indexing, for instance, treads correspond

to different term frequencies. Since there are only a finite number of document lengths in a given collection, and since for each length only a few term counts are possible, the function must necessarily be discontinuous. Actual posting lists, moreover, are samples from this space that reflect the distribution of text sizes in the collection and the distribution of term occurrences within text. Thus they have pronounced stairstep shapes, particularly in the central region corresponding to the most likely document sizes and term counts.

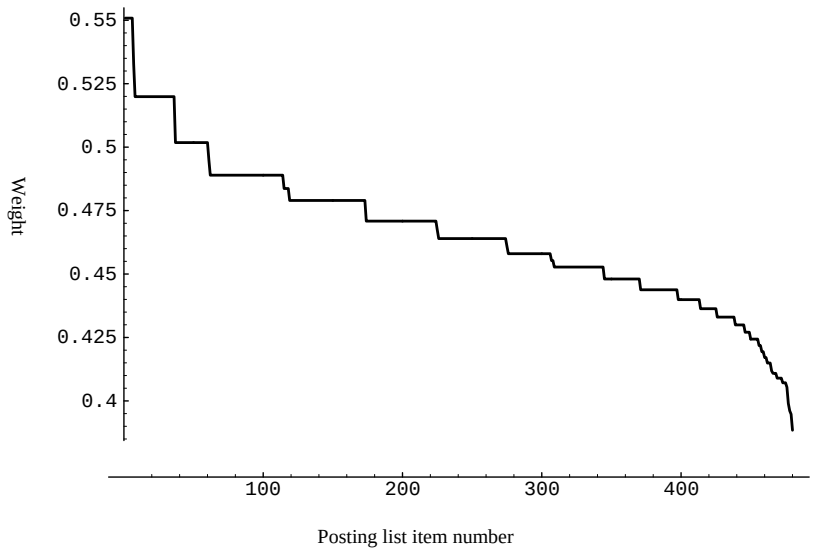


Figure 4.1: An actual posting list attenuation function. The posting list for "cat" among titles in the Virginia Tech library collection has 480 elements ranging in weight from 0.55 to 0.39. The weighting function is a variant of TF-IDF, so each stair tread corresponds to a particular combination of term count (almost always 1) and document length. The sigmoid shape is typical of a TF weighting function: the middle weights, corresponding to the most likely term counts in the most likely text sizes, have the longest treads, while the high and low weights, corresponding to very short and long texts, are represented by only a few objects.

The shape of a typical posting list curve depends on the weighting system used. Any monotonically non-increasing function can theoretically be achieved by manipulating the way weights are assigned to the objects during indexing. Indexing systems based on document characteristics, however, will produce curves that reflect the statistical character of the document collection. For instance, assume that document text sizes are normally distributed, and that any posting list is a random sample drawn from that distribution.

To facilitate statistical analysis of our model, we envision a text collection as if all the texts

were strung together in some arbitrary order to form one long sequence of tokens. Each token is then considered a single event in a long event sequence, and the particular term used to index the token considered the outcome of the event. Basically, if there are T terms in the underlying language, we consider each token to be the result of a T-way choice, where the T different outcomes occur with widely disparate probabilities.

This is the model proposed by Shannon [SHANNON41]. It applies to growing document collections as well as it does to static collections. In Shannon's terminology, the document collection is a stream of tokens generated by an information source. The information source chooses terms from a language using criteria unknown to us, but following a definite probability distribution. We do not know the distribution *a priori,* but as the collection grows – as we get a larger and lager sample of the source's behavior – we can make better and better guesses of the generating distribution. In the case of text collections, no sample includes the full language; larger samples generally include terms never seen before, so that as a collection grows we gain information both about the underlying set of terms and the probability distribution within the set. It is a measure of the power of Shannon's model that this simple description covers so many situations so well.

If a text collection is a (relatively large) sample of an infinite stream of tokens, an individual document is a smaller sample from this same stream. The collection is partitioned into documents, each document being a contiguous sample of tokens (events) produced by the information source. Thus each document affords us a probability distribution that serves as a picture of the information source. The distance between this distribution and the distribution of the collection can be measured in a number of different ways [KAPUR92].

We assume that document lengths are normally distributed around some mean λ. This assumption fits both our intuitions and the VT Library collection data used in MARIAN. If we further assume that the occurrence of any particular term in Poisson – that later events with that term as outcome are not conditioned by earlier events – then we would expect the within-document distributions to be fairly close to the collection distribution. Recent research by Church [CHURCH95] has cast substantial doubt on the later assumption. In particular, Church hypothesizes that "interesting" terms – terms that reflect the content of the document – tend to deviate from Poisson production, while less significant terms continue to follow the Poisson model. While we find these results tantalizing, we will ignore them in this analysis, both because our dependence on a Poisson production model is low, and because preliminary checks indicate that non-Poisson terms are relatively rare.

These two assumptions allow us to predict the distribution of raw TF data – that is, pairs of $\langle ct_{Term}, ct_{Total} \rangle$ values – in the collection. In particular, if a term has a collection-wide frequency ϕ_t , it has an expected distance of ϕ_t^{-1} tokens between successive occurrences. The expected raw TF distribution reflects the interaction of variation in individual distances with variation in document lengths.

To take a specific example, choose a term t where $\phi_t^{-1} >> \lambda$, so that no document is expected to include more than one occurrence of t. This is the case for most terms in many collections. In a collection where λ=100, for instance, this applies to all terms that occur with a frequency substantially smaller than 0.01; that is, to all but a handful of terms in English text. When this condition obtains, we can ignore cases where $ct_{Term} > 1$, and the raw TF data degenerates to a simple [[DISCRETE-NORMAL]] distribution. Figure 4.2 shows the attenuation functions produced by several common weighting functions from this distribution of raw data.

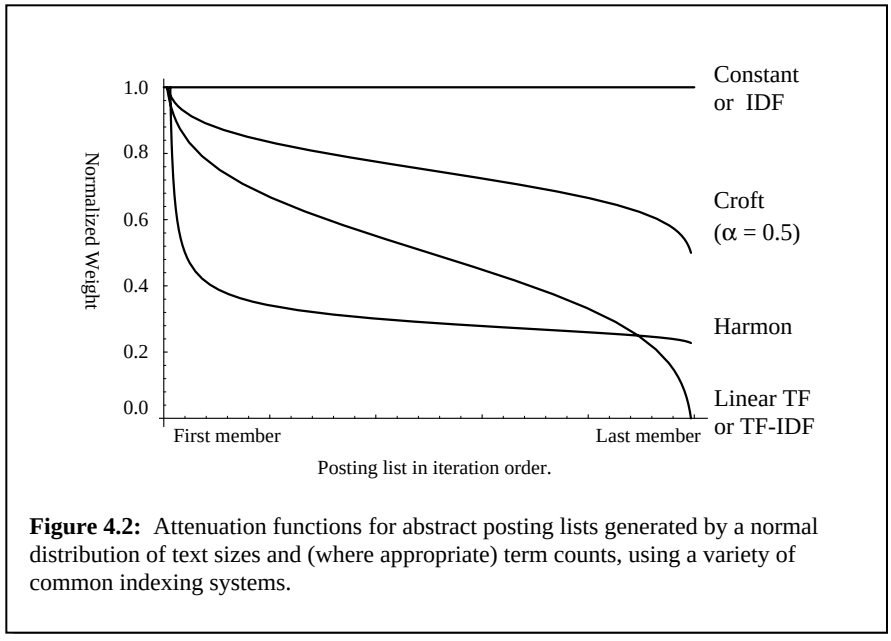


Figure 4.2: Attenuation functions for abstract posting lists generated by a normal distribution of text sizes and (where appropriate) term counts, using a variety of common indexing systems.

4.1.2 Ranked Retrieval as Weighted Object Set Union

A ranked retrieval system produces a weighted object set of all documents matching a given query, from those with the closest match at the top, down through any document that matches the query, however trivially. If the query is a flat or weighted collection of document

components (as opposed to a statement in some calculus) then any document matching any component will be part of the result set. In particular, if we pretend that both queries and documents are simple pieces of text, any document that matches any word in the query will appear in the result set. Thus we can model a search engine of this type as a weighted object set union.

Search engines covered by this model include all those based on the *vector model* [SALT&McG83], some probabilistic search engines [CROFT83], information theoretic engines [vanRIJSBERGEN79], and many ad hoc implementations. These search engines draw on divergent models of the retrieval process, use different means of assigning posting weights, and use different weight combining functions. They have in common that the final weight of an item in the union is a function of the weights obtaining between the terms in the query and the documents in the collection. In particular, they all implement a weight combining function based on a sum. In the case of the vector model, this sum is a (normalized or unnormalized) inner product; in the probabilistic model, a sum of partial contributions to the estimated relevance; in the information theoretic, a calculation of the relative entropy or cross entropy of the document. The component weights may be manipulated before being summed, and the final weight is often a function of the sum, but the sum remains the central operation.

The intuition underlying summing weights is straightforward. All other things being equal, a document that matches more parts of the query should receive a higher rank than one that matches fewer. Of course, all other things are not equal: different query components have different differentiating ability, and a match on an infrequent term in the language, for instance, may count more than a match on a frequent term. This disparity is the motivating force behind the use of stopping rules. Low-weight members of low-weight components, it is argued, can be reasonably discarded with little effect on the final weights of the components at the head of the combined set.

If the weighting function of a union set is based on a sum, we expect the resulting weight attenuation function to be made up of segments. If two weighted object sets are combined, for instance, we expect elements from the set overlap to have noticeably more weight than elements that occur in only one component set. If the attenuation functions of the component sets are sufficiently steep, some high-weight single elements may end up higher in the ordering than some overlap elements, but as as tendency, the two segments will be distinct. This effect is clear in practice. In Figure 4.3, for instance, we see summative unions of two, four, and 27 posting lists for terms from veterinary medicine. When two sets are summed, the distinction between the overlap segment and the single-set segment is very clear. As more sets are added, the distinction between segments becomes less and less well defined. The curve for the 27 set combination is, in fact, almost completely smooth. This is only to be expected. With 2^{27} different possible subsets of posting lists with widely varying IDF values, almost any gradation is possible.

In order to model the behavior of a summative union, we must first develop the statistics that determine overlaps among the component sets. When the component sets are posting lists, this is governed by the *hypergeometric distribution* (see Appendix). In this section we will only be interested in the expected value (mean) for the distribution. When we examine stopping rules, we will also be interested in the *quantile:* the first value for which the cumulative probability density exceeds some value.

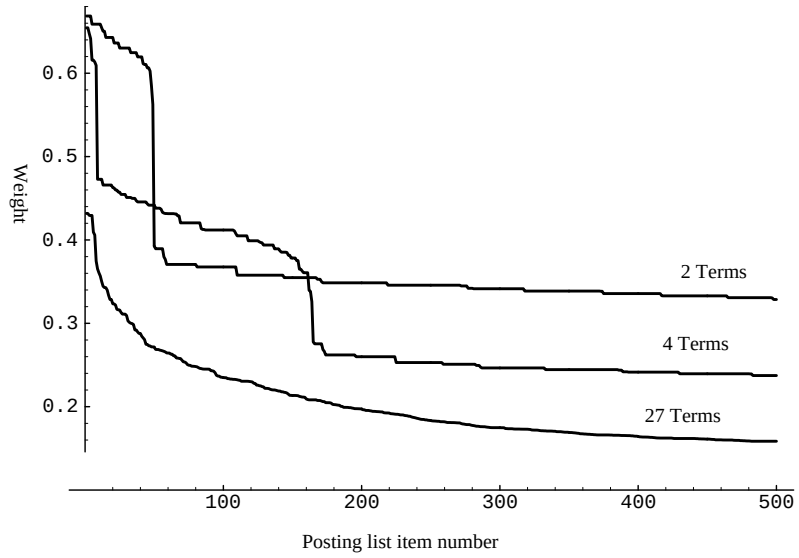


Figure 4.3: Initial sections of actual set attenuation functions for combined sets from the VT library title collection. In the first set, two terms ("cat" and "dog") produce a small high-weight area where the terms overlap, followed by a long tail. In the second, four terms ("cat", "dog", "anatomy" and "physiology") produce a tiny high-weight head, a moderate mid-weight section where two or three terms match, and then a long tail. A set of 27 terms from veterinary science produces a smoother curve with no apparent head.

When we take the union of two posting lists of size **n** and **m** in a universe of **N** items, we expect the size of their overlap to vary around the mean. In other words, we expect there to be

$$n + m - nm/N$$

unique items in the union, of which **nm/N** occur in both posting lists, **n-nm/N** occur in only the first, and **m-nm/N** occur in only the second (see Figure 4.4). With more component sets, we must consider all possible overlaps between pairs, then triples, and so on. To simplify matters, let us consider the case where all **k** component sets have a single size **n**. Let us further agree that each of the component sets have a constant weight attenuation function, and that the weight combining function of the union be a simple weighted sum, or average. That way the weight of any element in the union will be directly proportional to the number of component sets in which it occurs. The mean number of items found in all **k** sets will be

$$\text{overlap}_k = \frac{n^k}{N^{k-1}},$$

the mean number found in (k-1) sets will be

$$\text{overlap}_{k-1} = \binom{k}{k-1} \frac{n^{k-1}}{N^{k-2}} - \text{overlap}_k \quad ,$$

and in general, the mean number found in i sets ($1 \le i \le k$) will be

$$\text{overlap}_i = \binom{k}{i} \frac{n^{i}}{N^{i-1}} - \text{overlap}_{i+1} \quad .$$

Overlap size thus drops off geometrically as i increases. We expect no overlap at all to occur until $n^k \ge N$, no overlap of three sets to occur until $n^k \ge N^2$, and so forth. This is illustrated by the six-set case in Figure 4.4, and is consistent with the actual unions shown in Figure 4.3.

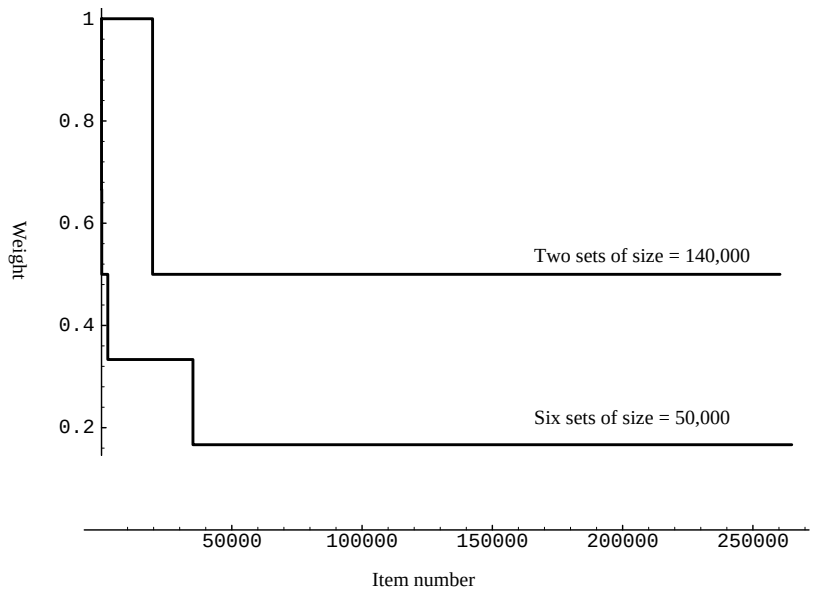


Figure 4.4: Expected shape of the attenuation functions for simple summative unions of two and six weighted object sets of equal size and constant weight (= 1). Collection size in both cases is 1,000,000, but the size of the component sets have been manipulated to produce coextensive curves. Note that overlaps of greater than three out of six sets have vanishing expected sizes. With these parameters, expected size of an overlap of all six sets is 0; of five, 2; and of four, 92. This is a fair number of documents for a user to examine, but miniscule compared to the total number of expected items in the union (264,908).

4.2 Opportunistic Search and Stopping Rules

Suppose that we are constructing the union of **k** weighted object sets using a simple or

normalized sum as our combining function. We want to formulate our construction in such a way that we can stop before reaching the end of some or all of the component sets, if we find at some point that we have already established the initial segment of the union well enough to meet some heuristic criteria. This is what we mean by opportunistic search, and the criteria are our stopping rules.

During this process, there are two structures of interest: the collection of component sets and the set of items that we have already seen, which we will refer to as the partial set. In a real search engine, the first is likely to be a set of disk files; the second, some form of accumulator bank [HARMAN92]. During the search process, items are run from the collection into the partial set. If an item is not already in the partial set, it is added; if it is, its weight is added to that of already accumulated. At any point in this process, the partial set is an approximation of final union set in two ways. Some items only occur in the unexplored portion of the component set collection, and are thus missing from the partial set. And any item in the partial set that also occurs in the unexplored collection has a weight lower than its ideal. As the exploration continues and the unexplored portion shrinks, this approximation gets better; when it gets good enough, we stop.

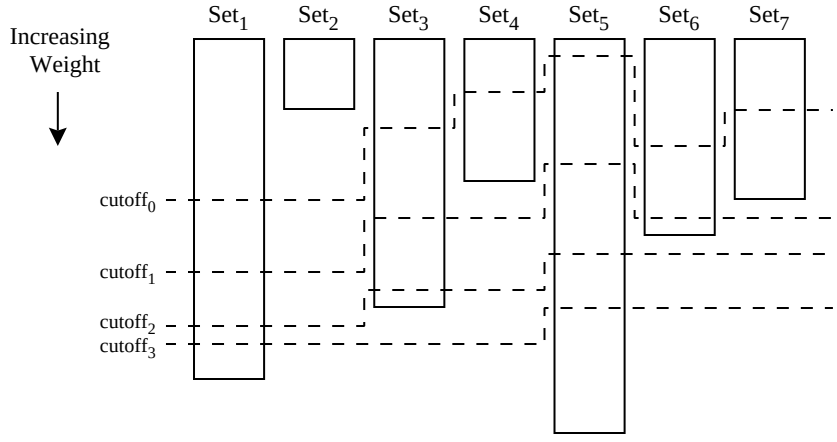


Figure 4.5: During opportunistic search, the MARIAN search engine explores a component set collection through regions of comparable weights. In each iteration, a cutoff weight is established and the sets are explored up to the frontier established by that weight.

Much attention has been placed in stopping rules research on the order of exploration of the component collection. Early systems [SM&vRIJS81, BUCK&LEW85] explored each list completely before going on to the next. Wong and Lee refined this by exploring each list a page

at a time, always choosing the page in the unexplored portion with the highest initial posting weight as the next to run into the partial set. The MARIAN opportunistic search engine takes this process to its logical conclusion and explores the lists in a fully interleaved manner. (Paging is handled at a lower implementation level.) At each stage in the search, the engine draws a frontier through the entire component collection, all members beyond which have a weight below some cut-off value (Figure 4.5). The first cut-off is set quite low so that a good initial approximation can be made. Subsequent cut-offs are derived as a side-effect of applying the stopping rules. This order is easy to deal with in the abstract, so we will use it in our discussion.

When a new frontier has been reached and all the items encountered fed into the partial set, the top portion of the partial set is considered. There are two questions we can ask of this portion:

- Is the composition likely to change?
- Is the order likely to change?

The composition can change if there is an item in the unexplored portion of the set whose total accrued weight would be greater than the lowest weight in the initial portion, or if there is an item in the remainder of the partial set that would be promoted into the top portion by a "hit" from the unexplored component collection. The order can change if a "hit" occurs within the initial portion. In both cases, we can define "likely" to mean "within some range of probabilities." A stopping rule makes a guess at how likely one of these events are to occur, and terminates the search if the probability is below some acceptable level.

4.3 Analysis: When Can We Stop?

In this section we evaluate the stopping rules defined above by examining the probabilities that exploring the rest of the way through a component collection will affect the top portion of a partial set. This can be broken into three questions: what is the probability that one or more elements discovered in the exploration will change the order of elements in the top portion; what is the probability that a combination of weights from the unexplored collection will cause a previously unseen object to invade the top portion; and what is the probability that one or more elements discovered in the exploration will promote an element from the lower portion of the partial set to the upper?

The first question can be re-cast simply as a question of sampling. Consider the top **t**

elements of the partial set to be a sample (of size **t**) drawn without replacement from the set of all **N** documents. The unexplored portion of the component set collection is a sample (of size **c**) drawn from the same collection *with* replacement. We are interested in the probability that these two samples have an element in common.* For the moment, we will ignore the internal structure of the unexplored collection, and consider only the number of unique elements in it. We can use the hypergeometric distribution $h_{<N, t, c>}$ to calculate the chance that one hits the top portion.

Say we want to know whether the top 20 elements of a partial union set are stable at the .01 level; that is, whether there is a 99% chance that they will be unaffected by exploring the remainder of the component collection. Let us assume a universe of 1,000,000 objects and ask how small the unexplored collection needs to be to obtain this level of stability. We want to find a **c** for which

$$h_{<1000000, 20, c>}(0) \geq .99$$

Using the special case for x=0 (see Appendix), we find that the largest **c** for which the inequality holds is 502.

Remember that we are dealing here with posting lists of thousands, even tens of thousands of elements, and posting list collections of hundreds of thousands of elements. It is these numbers that drive us to use stopping rule methods. Once we have reduced such a collection to the point where only 500 elements remain, we are virtually done, and might as well load in the last few. The amount of computation time that has been spent in calculating probabilities and maintaining the extra structure needed for opportunistic search is already more than would be required to run 500 elements into an accumulator bank. Thus we conclude that to satisfy this criterion of stability we are best off doing an exhaustive exploration of the component set collection.

***** To be completely precise, we are interested in the probability that these two samples have an element in common where the total weight of the element in the unexplored collection is greater than the difference in weight between the element in the partial set and the element immediately above it: the probability that we will get a hit that displaces the item hit. In point of fact, however, the weight differences between neighboring items in a union set tend to be very small compared to item weights in the component sets (compare Figure 4.3). When the item is on a stair tread, there is no difference between neighboring elements. So we are not far off in asserting that *any* hit in the initial segment will displace the element hit.

Fortunately, there is ample justification for relaxing this criterion. Consider the case of a simple text search, where a user is attempting to match a vector of terms across a text collection. No known ranking function does a very good job of ordering the results of such a search, although many correlate well over a large sample with users' judgements (for an overview of related research, see [SARACEVIC91]). So we may well believe that we can give up precise ranking in the set *if we can be certain that the best documents are in the top portion of the set.* In other words, if we have established the content of the top portion correctly, we can afford to be relaxed about the order.

It is one of the features of the hypergeometric distribution that even for large N only small sets are at all likely to be disjoint. Figure 4.6, for instance, shows the probability that two samples drawn from a universe of 10,000 events will be disjoint. As can be seen, unless at least one sample is very small (less than a few hundred elements) there is virtually no chance that the pair will be disjoint. This "handball court" shape is less pronounced with smaller total universes, but the effect is the same: unless the samples are extremely small, the chances are that they will have some overlap.

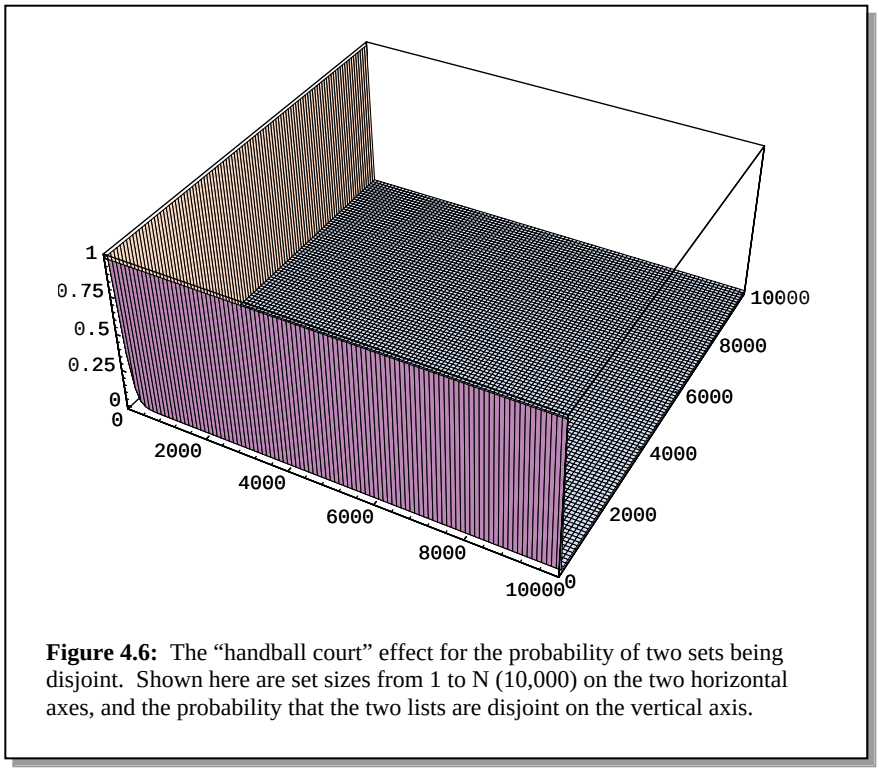


Figure 4.6: The "handball court" effect for the probability of two sets being disjoint. Shown here are set sizes from 1 to N (10,000) on the two horizontal axes, and the probability that the two lists are disjoint on the vertical axis.

The second of our tests asks whether any elements of the unexplored portion of the component set collection are likely to belong in the top portion of the combined set. We presume that this can only happen for elements that occur in more than one among component set. After all, we have already transferred a large number of elements of higher weight than those remaining. It is easy to test whether any combination(s) of expected weights from unexhausted component sets would be greater than the least weight in the head of the partial set. If so, it remains to calculate the expected size of those overlaps. This is a special case of the calculation for the third test, and any exploration that passes the third will pass the second.

The third test depends on the probability of a promotion from the lower portion of the partial set into the top portion. We want to know whether the dividing line between these two sections of the union is **stable** in the following sense:

> **Definition:** A point in the partial set is *stable* if none of the items below that point are likely to be promoted above it during exploration of the remainder of the component set collection. In particular, a point is *stable at a given level* if the expected probability of such a promoting hit is less than that level.

Note that a stable point is only that: a point. A point a few elements to one side or the other may not be stable, particularly if the stable point is on a stairstep riser. This parallels our discussion of the first test above. Just because a section of the combined list is stable – unlikely to receive any new elements during the remainder of the exploration – does not mean that it is static – that elements within the section are not likely to be reordered by further exploration.

We break the third test into a sequence of tests. First, we consider single hits from the remainder of the component set collection, and ask how many of these are likely to move an element from the tail of the partial set into the head. This is a sampling question, but not simply a question of the probability of overlap between the unexplored collection and the tail. It may well be the case that most of the weights in both the unexplored portion and the combined tail are very low, much smaller than our candidate point of stability. We are concerned only with a subspace of all the possible hits and misses: with only those hits whose combined weight is greater than the candidate point.

Next, we estimate how many objects in the unexplored collection occur in two lists, calculate an expected weight for them, and count potential promoting hits for those. Objects that occur in two component sets will have higher weight than those that only occur in one, so there

will be more items in the tail that may be promoted. On the other hand, there will be fewer objects that occur twice in the unexplored collection. We continue by estimating the number of objects in three component sets, and so forth.

Practically speaking, we generally only need to ask the question for items from single component sets and from pairs of sets. As mentioned in Section 4.2, the expected size of overlap segments decrease geometrically as the number of sets involved goes up. Suppose that P_{cutoff}, the certainty that we want to achieve, is .999. (In real applications, we would never be this precise. The error in our indexing functions is already greater than .001; a more realistic level is .9 or .95.) If the component sets are posting lists for terms in ordinary language, the largest we can expect them to get is in the range 0.01N–0.1N. Even at the largest values, it is barely possible that we should need to consider triples of sets; certainly we need consider nothing less common. MARIAN opportunistic search uses two heuristics:

- Assume that all the items remaining in the unexplored portion are unique items. Count the number of potential promotions in the tail of the partial set. What are the chances that these two intersect?
- Calculate the expected number of items in the unexplored collection that occur in two lists. Assume that any of these will promote any item in the tail. What are the chances of a hit?

If both these probabilities are sufficiently low, the search engine can stop the exploration. These are cost-effective approximations to the more rigorous sequence of calculations in this paper.

The first calculation in our sequence is of the probability that the expected weight of a single unseen item will promote any item in the partial set's tail. In particular, if there are **N** possible items in the collection as a whole, $\mathbf{c_1}$ items that occur only once in the unexplored collection and **d** items in the tail with weight greater than the difference between the expected weight and the least weight in the top portion, we want to know if

$$h_{<N,\, c_1,\, d>}(0) > P_{cutoff}.$$

As before, we are only interested in opportunistic search when **c** is quite large. $\mathbf{c_1}$ is smaller than **c**, but not appreciably. This means that we can only stop when **d** is quite small (see Figure 4.7). In other words, the only places on the partial set attenuation curve where this condition is met are those where the average slope is very large: where one can achieve a large decrease in weight over very few elements. When we are combining only a few sets, the union attenuation function is still primarily a stairstep function (see Figure 4.3). In this case, we are only likely to be able to stop when our candidate stable point is close to the trailing edge of a stair

tread. Such stable points do exist, but they are unlikely to exist at predetermined points. In particular, if we are trying to establish stability of the first element, or the tenth or twentieth, we are probably out of luck.

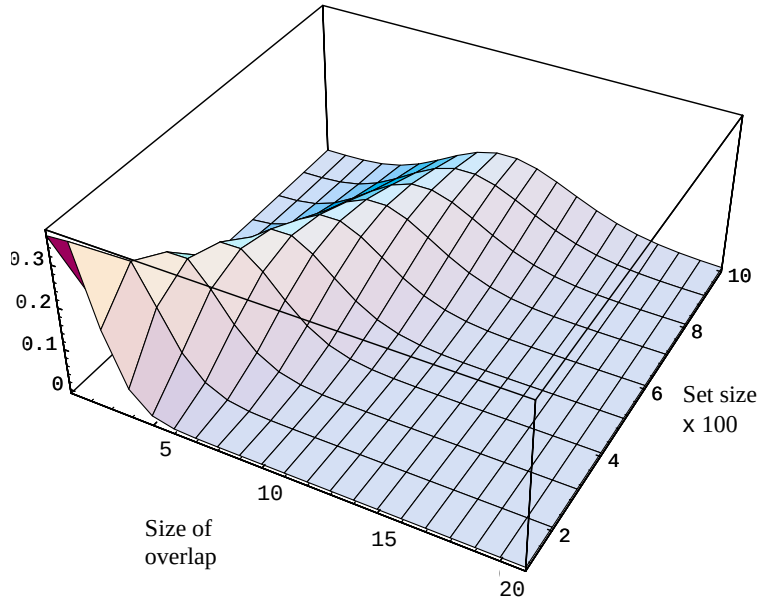


Figure 4.7: Probability density surface for overlaps between a 1000-element set and sets of sizes from 100 to 1000 in a universe of 100,000 items. For the case of $h_{<100000,\ 1000,\ 100>}$, overlaps of 0 and 1 are most likely. By $h_{<100000,\ 1000,\ 1000>}$, the mean overlap is 10 items. Even in this case, though, the probability of an overlap of size 20 or greater is still less than .002.

When many sets are combined in an opportunistic union, the situation is more encouraging. In the 27-set union from Figure 4.3, for instance, there is a large initial region where the average slope is uniformly high. Any candidate point in this region has the required characteristic that a single hit from one of the 27 lists will promote very few elements below it. In the later region of the attenuation curve, this is no longer true, but this region is unlikely to be explored by any human user (see Figure 4.8).

A similar analysis applies for objects that occur in several of the component sets. If an item occurs in two or more component sets, then we need to know if the sum of the expected weights for those sets is likely to cause a promotion. Since the total hit weight is higher, we will need to include a larger portion of the tail in our one sample; however, the expected number of items in

common between component sets is likely to be much lower. The hypergeometric distribution again gives us a maximum number of acceptable promotions. Together, the numbers define a second line that the curve must also fall below (see Figure 4.8). This condition may be either more stringent or less than the first. In the normal case for ranked retrieval, where the number of component sets is small and their sizes less than 0.1N, the expected overlap will generally be an order of magnitude smaller than the expected number of single hits. The change of weight, on the other hand, is comparable. For a simple summative union, it is in fact twice the expected weight for a single unseen item. So any situation satisfying the test for single hits will pass the test for multiple hits.

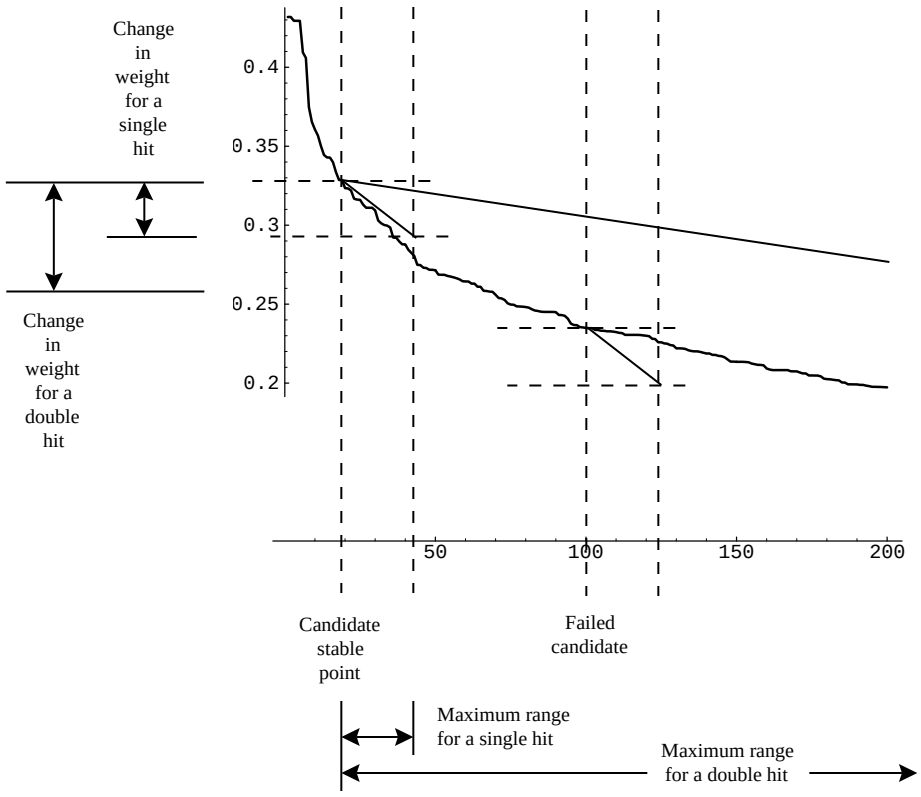


Figure 4.8: In this magnification of the initial segment of the 27-set union from Figure 4.3, two candidate stable points are shown. The candidate at item #20 meets the abstract stopping rules for stability, as fewer points below it in the union iteration would be promoted by a single hit than the maximum number. The point at item #100 fails the same test. Also shown is the test for pairs of hits. Note that while the change in weight for this test is only twice that for the single hit test, the maximum acceptable range is many times greater.

This situation only reverses in two cases. One case – when component set sizes approach

the collection size – does not occur in text retrieval. The other case is when large numbers of component sets remain unexplored. In this case, the test for multiple hits becomes more stringent than that for single hits, simply because the number of multiple hits is on the same order as the number of single hits.

4.4 Discussion

The preceding analysis presumes statistical independence of the component weighted object sets. This is a common assumption – and in fact the only assumption that allows a tractable analysis – but in point of fact it is completely mistaken. Words in English have complex networks of interdependency [DEER&al90]. Words in a query are if anything less independent. We might say that users construct queries in the fervent hope that their search terms are *not* independent; that they are likely to co-occur in documents of interest.

Suppose we are searching a moderate-size research library collection (the Virginia Tech collection of 1994, with a total of 960,000 volumes) using the query **Author: Asimov; Title: foundation**. In the Virginia Tech collection there are two authors named Asimov: Isaac, with a total of 137 books to his credit, and Janet with two. The word "foundation" in the title retrieves 3754 books, including nine by Asimov. Let us pretend that these two keys are statistically independent and calculate the odds of any overlap at all.

As developed above, the odds of an overlap of x works are

$$h_{<900000,\ 139,\ 3574>}(x) \quad 0 \le x \le 139$$

With these parameters, the odds are 0.56 that the two sets have no intersection at all. Above x=3 the odds become miniscule, and the probability of an overlap of nine works calculates to 9.00×10^{-9}. Such an event thus satisfies the most stringent tests for statistical significance – not surprisingly, since there is in fact a significant correlation in the real world between Isaac Asimov and the word "Foundation." It is that correlation that informed the query, and that undermines any statistical rules for curtailing exploration.

This does not mean that stopping rules are useless. We noted above that the rules favor situations with large numbers of component sets. An opportunistic search on the 27-term query in Figure 4.3, for example, stops relatively early in the process. In applications such as clustering and relevance feedback, where full-size documents are used as queries, opportunistic search can make a difference. Typical queries authored by users, on the other hand, tend to be short and to

include fewer high-frequency terms than ordinary unconstrained language. These queries invoke precisely the statistical patterns that mitigate against stable points in the upper portion of the union. So it is no surprise that when user-supplied queries are run against large document collections, stopping rules fail to stop.

4.5 Acknowledgements

This paper and the MARIAN system both owe much to Dr. E.A. Fox. Ed conceived of MARIAN as a production system, and has championed it though a long and complex development process. He also is responsible for introducing me to the method of stopping rules, in particular to the research of Wong and Lee, and for recommending that we include it in MARIAN. He has been extremely encouraging throughout this research; I hope that he does not find the results herein too *dis*couraging.

Both the Virginia Tech Computing Center and the Tech Library have supported MARIAN; the Library by selflessly sharing their data, and the Computing Center by funding the development process. Too many individuals in the Center and in the Department of Computer Science have contributed to MARIAN to mention here. I will make a single exception and thank Dr. Ben Cline, without whom the effort would never have gotten off the ground.

Mike Keenan pointed out to me that the hypergeometric distribution was "the one I was looking for." My thanks go to him and to the rest of the Dead Logicians Society for sitting through earlier presentations of this research and making valuable suggestions.

4.6 References

[BUCK&LEW85] Buckley, Chris and Alan F. Lewit, "Optimization of inverted vector searches." *Eighth Int'l. Conf. on Research & Development in Information Retrieval (Montreal,1985).* 97-110.

[CROFT83] Croft, W. Bruce, "Experiments with representation in a document retrieval system." *Information Technology: Research & Development* **2**:1 (1983) 1-21.

[DEER&al90] Deerwester, Scott, et al., "Indexing by latent semantic analysis." *JASIS* **41**:6 (1990) 391-407.

[FOXetal93] Fox, E.A. et al., "Development of a modern OPAC: from REVTOLC to MARIAN." *Proc. SIGIR '93: 16th Int'l. Conf. on Research & Development in Information Retrieval (Pittsburgh, PA, 1993).* 248-259.

[HARMON92] Harmon, Donna, "Ranking algorithms." In Frakes, W.B. and R. Baeza-Yates, Eds, *Information*

Retrieval: Data Structures and Algorithms. New Jersey: Prentice-Hall, 1992. 363-392.

[HARM&CAND90] Harmon, Donna and Gerald Candela, "Retrieving records from a gigabyte of text on a minicomputer using statistical ranking." *JASIS* **41**:8 (1990) 581-589.

[LEE&WONG91] Lee, Dik Lun and Wai Yee Peter Wong, "Partial document ranking by heuristic methods." *Proc. Int'l. Conf. on Computing and Information (Ottawa, May 27-29, 1991)* 231-239.

[LUCARELLA88] Lucarella, Dario, "A Document retrieval system based on nearest neighbor searching." *Journal of Information Science* **14** (1988) 25-33.

[RASMUSSEN92] Rasmussen, Edie, "Clustering algorithms." In Frakes, W.B. and R. Baeza-Yates, Eds, *Information Retrieval: Data Structures and Algorithms.* New Jersey: Prentice-Hall, 1992. 419-442.

[SALT&BUCK88] Salton, Gerard and Chris Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management* **24**:5 (1988) 513-523.

[SALT&McG83] Salton, Gerard and M. McGill, *Introduction to Modern Information Retrieval.* New York: McGraw-Hill, 1983.

[SARACEVIC91] Saracevic, Tefko, "Individual differences in organizing, searching and retrieving information." *Proc. ASIS '91 (Washington, DC, Oct 27-31, 1991).* 82-86.

[SM&vRIJS81] Smeaton, A.F. and C.J. van Rijsbergen, "The Nearest neighbor problem in information retrieval: an algorithm using upperbounds." *Proc. Int'l. Conf. on Information Storage and Retrieval (SIGIR) (Oakland, CA, 1981)* ACM. 83-87.

[vanRIJSBERGEN79] van Rijsbergen, C.J. *Information Retrieval* 2nd Ed, London: Butterworths, 1979. See esp. pp. 130-139.

[WONG&LEE91] Wong, Wai Yee Peter and Dik Lun Lee, "Implementations of partial document ranking using inverted files." Unpublished report, August 1991.

4.7 Appendix: The Hypergeometric Distribution

The *hypergeometric distribution* is a discrete probability distribution constructed by sampling a universe of N items without replacement. It is usually explained as the distribution of probabilities among the number of successes in a sample of n items, where m of the N items in the universe count as successes. For our analysis, however, it is more intuitive to use an equivalent formulation based on the size of the overlap between two subsets. In this formulation, the hypergeometric distribution $\mathbf{h_{<N, n, m>}}$ is the distribution of probabilities among all possible sizes of the overlap of two sets of sizes **n** and **m** in a universe of **N** items. The distribution is defined over all possible sizes of the intersection, from 0 to the the lesser of **n** and **m**. We will use the notation $h_{<N, n, m>}(x)$, $0 \le x \le \min(n, m)$, to denote the probability of the two subsets sharing exactly **x** items. That probability is given by the equation:

$$h_{<N, n, m>}(x) = \frac{\binom{m}{x}\binom{N-m}{n-x}}{\binom{N}{n}}$$

$$= \frac{n! \cdot m! \cdot (N-n)! \cdot (N-m)!}{x! \cdot (n-x)! \cdot (m-x)! \cdot N! \cdot (N-n-m+x)!}$$

We are often interested in the probability that two subsets are disjoint; that they have no overlap. This value is given by

$$h_{<N,\, n,\, m>}(0) = \frac{\binom{N-m}{n}}{\binom{N}{n}}$$

$$= \frac{(N-n)! \cdot (N-m)!}{N! \cdot (N-n-m+x)!}$$

Computationally, the last formula looks like m factors (or n factors; we choose the smaller number for obvious reasons) of a steadily decreasing fraction:

$$= \frac{(N-n) \cdot (N-1) \cdot \ ... \ \cdot (N-n-m+1)}{N \cdot (n-1) \cdot \ ... \ \cdot (N-m+1)}$$

$$\cong \left(\frac{(N-n)}{N} \right)^{m}$$

Where this last, of course, is the value for the binomial distribution obtained by taking two samples *with replacement* from a universe of N items. The larger N is with respect to n and m, the better the binomial distribution approximates the hypergeometric.

The binomial and hypergeometric distributions are intimately related. They have the same mean:

$$\mu = \frac{n \cdot m}{N} \ .$$

The hypergeometric distribution, however, forms a sharper peak than the binomial distribution, growing sharper still as the sample sizes grow. Where the variance of the binomial distribution is given by $\sigma^2 = n \cdot (m/N) \cdot (1 - m/N)$, the variance of the hypergeometric distribution is

$$\sigma^2 = \frac{n \cdot m}{N-1} \left(1 - \frac{n}{N} \right) \left(1 - \frac{m}{N} \right) \ .$$

Chapter 5: Building a Unified Digital Library for Theses and Dissertations

5.0 Introduction

NDLTD is an international federation of member universities and other institutional members who wish to improve graduate education, promote access to scholarly research, increase sharing of knowledge, help universities build their information infrastructure, and extend the beneficial impact of digital libraries. More than 120 members are participating in the group. We consider two members of NDLTD. Virginia Tech was the first university in the United States to require all graduate students to submit theses and dissertations electronically. We illustrate search and presentation in the resulting VT-ETD collection. PhysDis is a discipline-based member of NDLTD which collects dissertations on physics from universities in Germany and beyond. The collection is disseminated using the Harvest™ system, which uses topic-specific agents to collect and extract indexing data from individual university collections. The agents extract summaries of content from harvested sources into a specific proprietary format (SOIF), which is used as both a metadata language and a transport format.

OAI provides an infrastructure for interoperability among sites supporting author self-archiving. It uses a standard data harvesting protocol building upon the Santa Fe Convention. The OAI protocol makes it possible to collect information from many types of repositories. We demonstrate OAI harvesting on the VT-ETD and PhysDis collections.

MARIAN is an indexing, search, and retrieval system optimized for digital libraries. It was developed at the Virginia Tech Computing Center, part of the Virginia Tech Information Systems. Recent research has focused on using the powerful representation system of MARIAN to span the differences in format and semantics within the NDLTD. We demonstrate a prototype of MARIAN that unifies the VT-ETD and PhysDis repositories into a single searchable union collection.

5.1 The Networked Digital Library of Theses and Dissertations

The Networked Digital library of Theses and Dissertations (NDLTD) (Figure 5.1) is an international federation of member universities and other institutional members who wish to improve graduate education, promote access to scholarly research, increase sharing of knowledge, help universities build their information infrastructure, and extend the beneficial impact of digital libraries. Currently, more than 120 members are participating in the group.

Because of heterogeneity among NDLTD members, interoperability is a major problem. NDLTD faces autonomy of management; decentralization; heterogeneity in all levels (metadata, protocols, repository technologies, language, character coding, nature of the data, user characteristics, preferences, and capabilities); and massive amounts of (dynamic) data. In this

presentation, we present the development of a global unified digital library of theses and dissertations. This system, which addresses many of the challenges above, is based on the MARIAN digital library system and aims to replace our simple federated search system (Figure 5.2) that has been running since late 1997 (see www.theses.org).

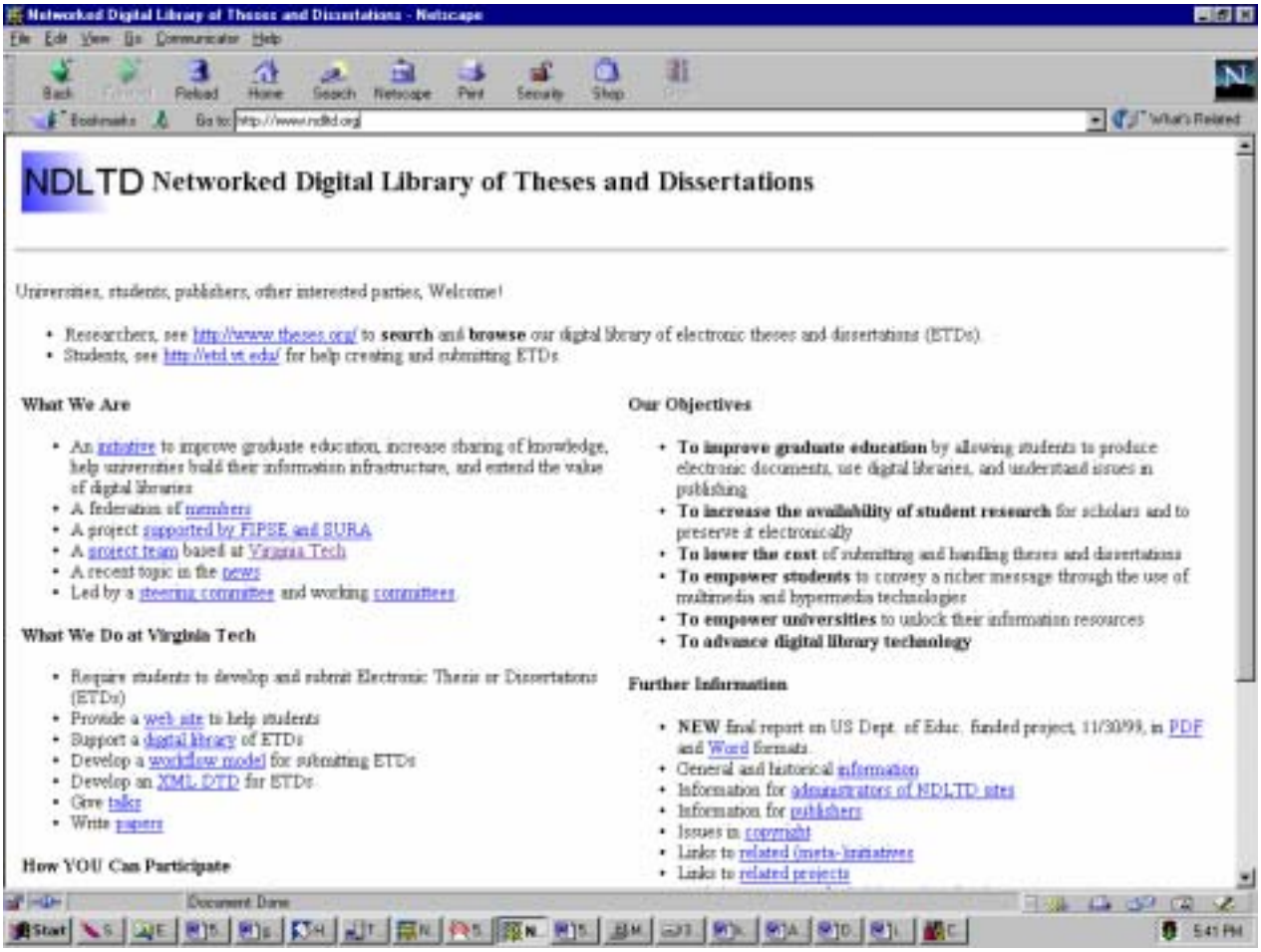


Figure 5.1. NDLTD Homepage

The current federated search system [Pow98] allows searching across several NDLTD members. Queries to the system are decomposed at run time into queries to those sources by using a simple description of the source capabilities. International sources are accessed through a simple dictionary-based multilingual translation of query terms. Data is not replicated and is guaranteed to be fresh at query time. However the current system has some major problems. The system translates queries in the level of CGI-based syntax, which is very volatile. Results from the several sources are not integrated into a unique ranked list and individual responses of each site are presented in the native user interface of the source (Figure 5.3). Performance is also a drawback. Such factors must be considered as network and server latency and availability, amount of data to be transferred, etc.

Figure 5.2. The homepage for the Federated Search System

The new system tries to solve those problems through an architecture based on harvesting mechanisms. The Marian provides a common query interface and integration platform. In our union archive based implementation, information from NDLTD members can be periodically extracted from different sources using several harvesting approaches (including the Harvest™ system and protocols such as Open Archives, Dienst, and Z39.50), processed (e.g., generation of indexes), merged with information from other sources, and then loaded into a centralized data store – the union archive. Queries are posed against the local data without further interaction with the original sources and modifications are filtered (for relevance, update-time) and used to update the union archive. Particularly, the use of the Open Archives protocol provides a partial solution for metadata interoperability problems and a simple but powerful harvesting mechanism to overcome heterogeneity barriers between NDLTD members. The new testbed system has started with two ETD collections: The collection of electronic theses and dissertations of Virginia Tech and a discipline based collection of Electronic theses and dissertations from PhysDis.

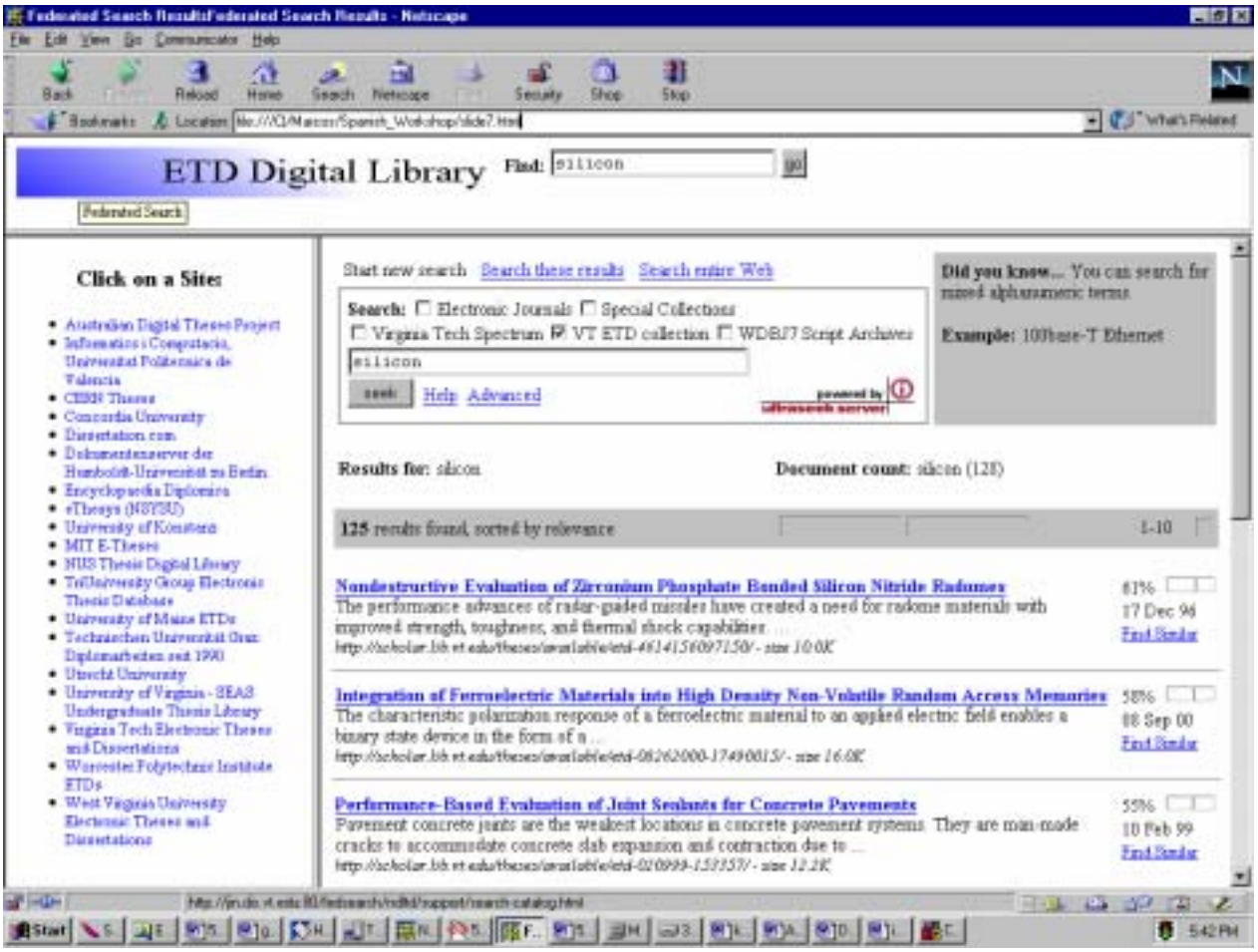


Figure 5.3. Presentation of the results of a query for Virginia Tech

Virginia Tech (http://etd.vt.edu) was the first university in the United States to require all graduate students to submit theses and dissertations electronically. PhysDis (http://elfikom.physik.uni-oldenburg.de/dissonline/PhysDis/_dis_europe.html) (Figure 5.4) is a discipline-based member of NDLTD which collects dissertations on physics from universities in Germany and beyond. The collection is disseminated using the Harvest™ system, which uses topic-specific agents to collect and extract indexing data from individual university collections. The agents extract summaries of content from harvested sources into a specific proprietary format (SOIF), which is used as both a metadata language and a transport format.

Figure 5.4. The PhysDis collection

5.2 Open Archives Initiative Interface

In order to combine the metadata from multiple sources like PhysDis and VTETD), we use a standard data harvesting protocol defined by the Open Archives Initiative (http://www.openarchives.org). Each archive is equipped with a software interface layer that understands specially formatted HTTP requests and returns the requested data using an XML transport format. As an example of this, we can issue the request to list all possible metadata formats supported by the archive (Figure 5.5).

The other services that may be requested are: the list of all partitions within the archive, the unique record numbers (Figure 5.6), the list of metadata formats in which a particular record may be retrieved, or the metadata associated with a particular record (Figure 5.7).

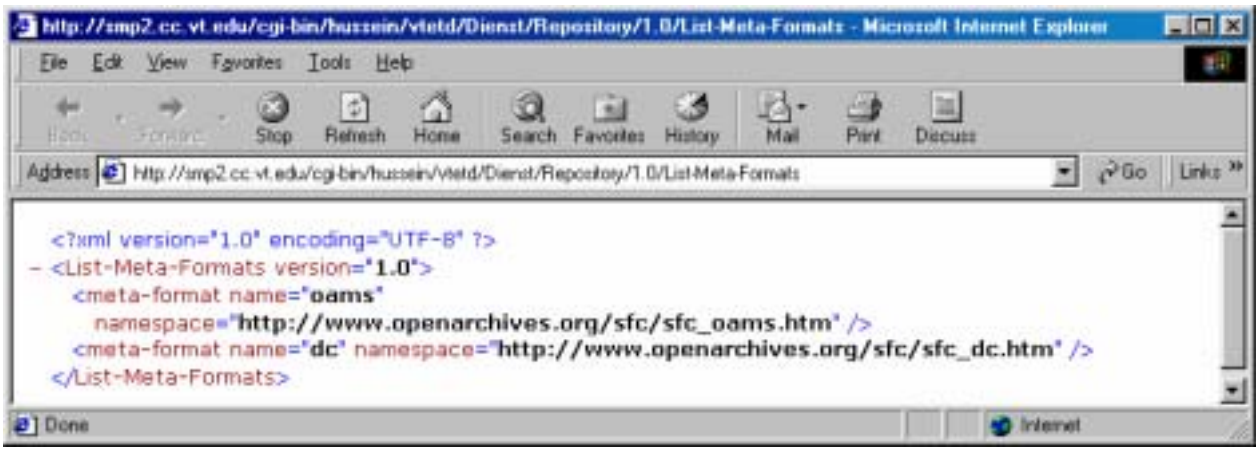


Figure 5.5. Request for metadata formats.

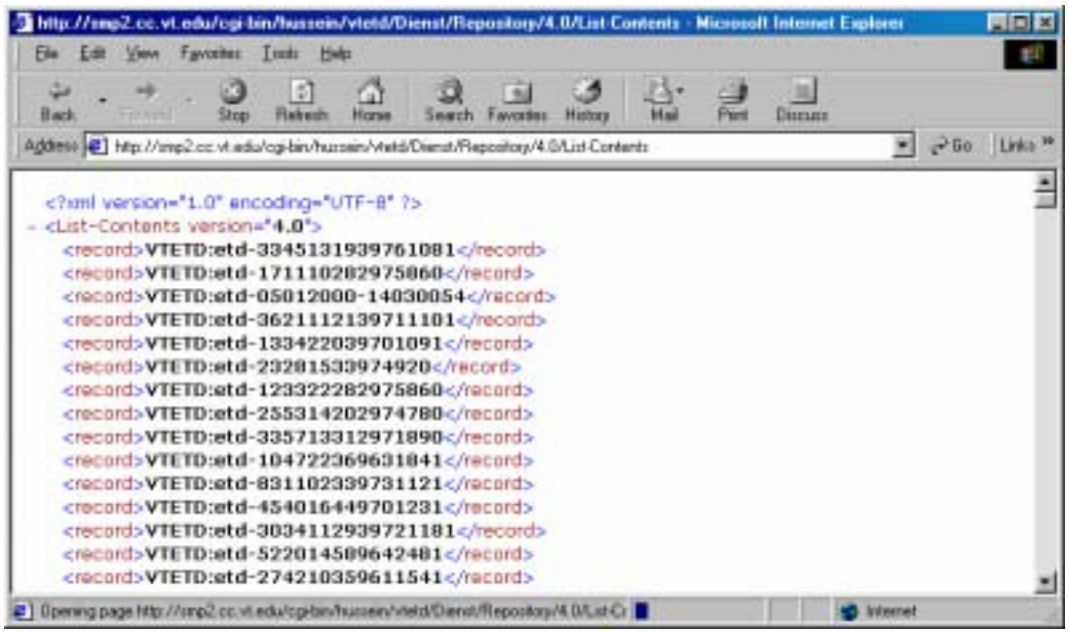


Figure 5.6. Request for list of record identifiers

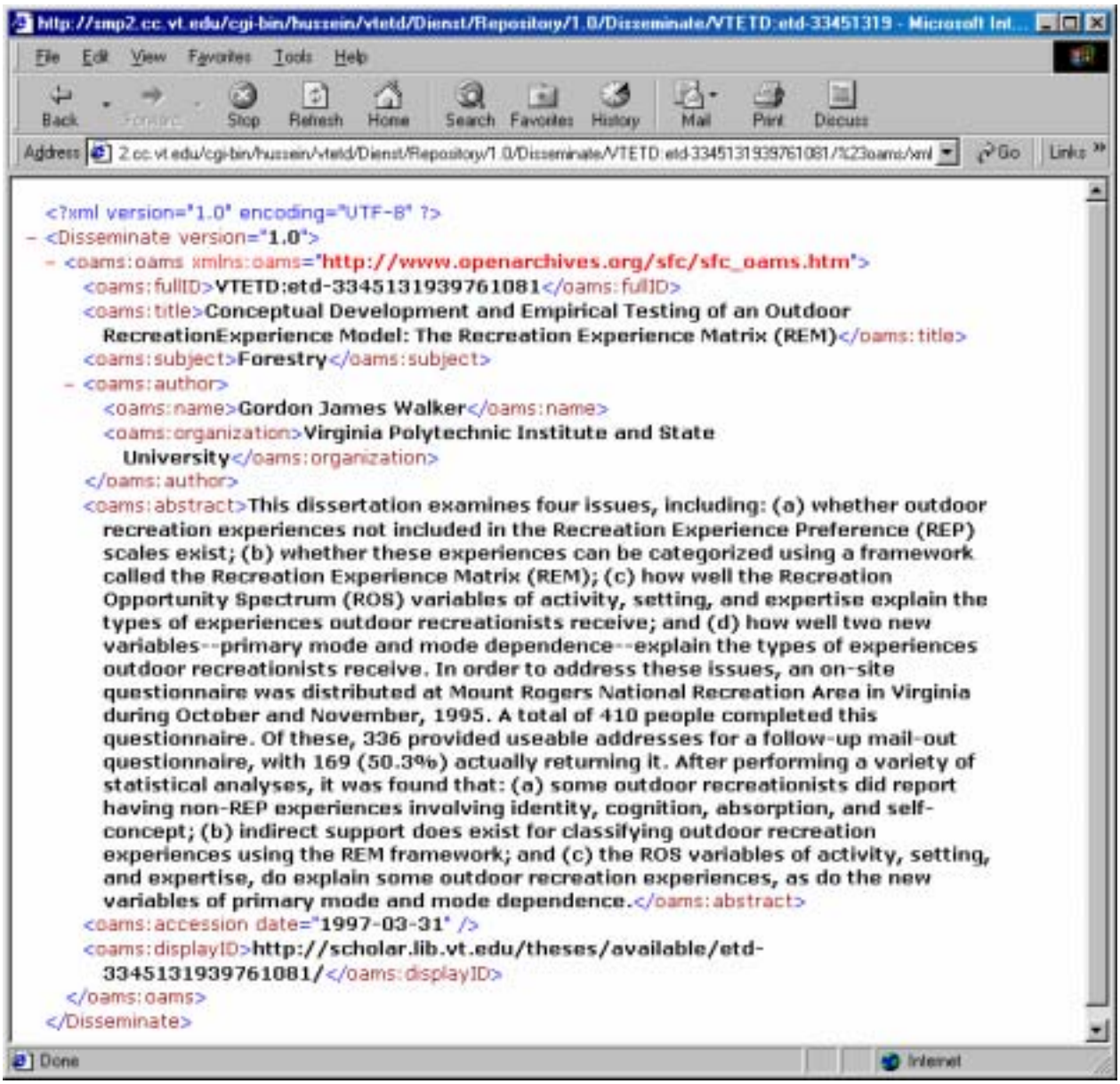


Figure 5.7. Request for metadata

Similarly, we can issue the same requests to the PhysDis collection and get responses in the same standard format.

We have developed software that implements this protocol in Perl and in Java. This code is freely available for others to use and modify as necessary to implement their own OAI interfaces. In addition we have developed a tool to check that an archive's implementation of the OAI protocol is correct. This tool, the Repository Explorer (Figure 5.8), allows a user to browse through the contents of a repository, while vigorously checking that the data returned by the server is properly formatted in response to every request.

Figure 5.8. Repository Explorer viewing metadata formats of VTETD collection.

Once a repository has a working OAI interface, any provider of services can harvest its data using the service requests supported by the OAI interface. One such service is the MARIAN Information Retrieval system that implements a search service across multiple archives.

5.3 Building a Union Collection from Harvested Repositories

Once data has been harvested from local repositories at NDLTD institutions, the second phase of interoperation begins. Structures within the harvested documents are examined. Salient points are revealed in the digital library schema to aid browsing and searching. Separate classes can be created for objects like authors that are independent of any given document. At times, these classes can create bridges to other classes of information in the library.

The harvested documents are analyzed and indexed, creating local images of the remote collections. The images all share a common vocabulary, but can differ greatly in structure. Remote sites that use a standard representation, like MARC or the evolving ETD metadata standard, are processed using the same transformations, but create separate images.

For some remote sites, federated searching may be a more attractive alternative than harvesting. These sites are represented as "opaque" document classes supporting remote search as a class method. Views and superclasses integrate the disparate structures into simpler synthetic classes. These classes provide access points to relevant portions of the unified library, including the most general aspects. The unified collection space is now ready for integrated services to be added.

5.4 Integrating the Union Collection for Unified Services

Integrated services to the unified Networked Digital Library of Theses and Dissertations are provided by the MARIAN system (Figure 5.9). A variety of services can be provided within the MARIAN architecture. This demonstration focuses on *search.* MARIAN can be adapted to work with one or several collections (Figure 5.10): the demonstrated system searches two NDLTD sites as a default. This default is part of the user's profile and can be changed to suit the individual. Documents from all sites in the union collection are returned, despite differences in format (Figure 5.11). Document descriptions are synthesized and displayed based on the locally cached data, but the authoritative version of the document is only a click away (Figure 5.12).

MARIAN search has several useful features. Descriptions of different aspects may be combined. For instance, we might want to know all works done under a certain major professor on a certain subject. It is also easy to search for the same description simultaneously in several fields. Many aspects of the search are under user control and can be personalized through preferences. For instance, we can modify how long MARIAN will wait for results from each collection. This allows us to simulate the workings of a federated search system.

In this way MARIAN gives us the tools to provide interoperability at the document format level and at the divide between federated search and harvesting. MARIAN views also provide interoperability at the service level, allowing us to present a unified view of diverse data to digital library users worldwide.

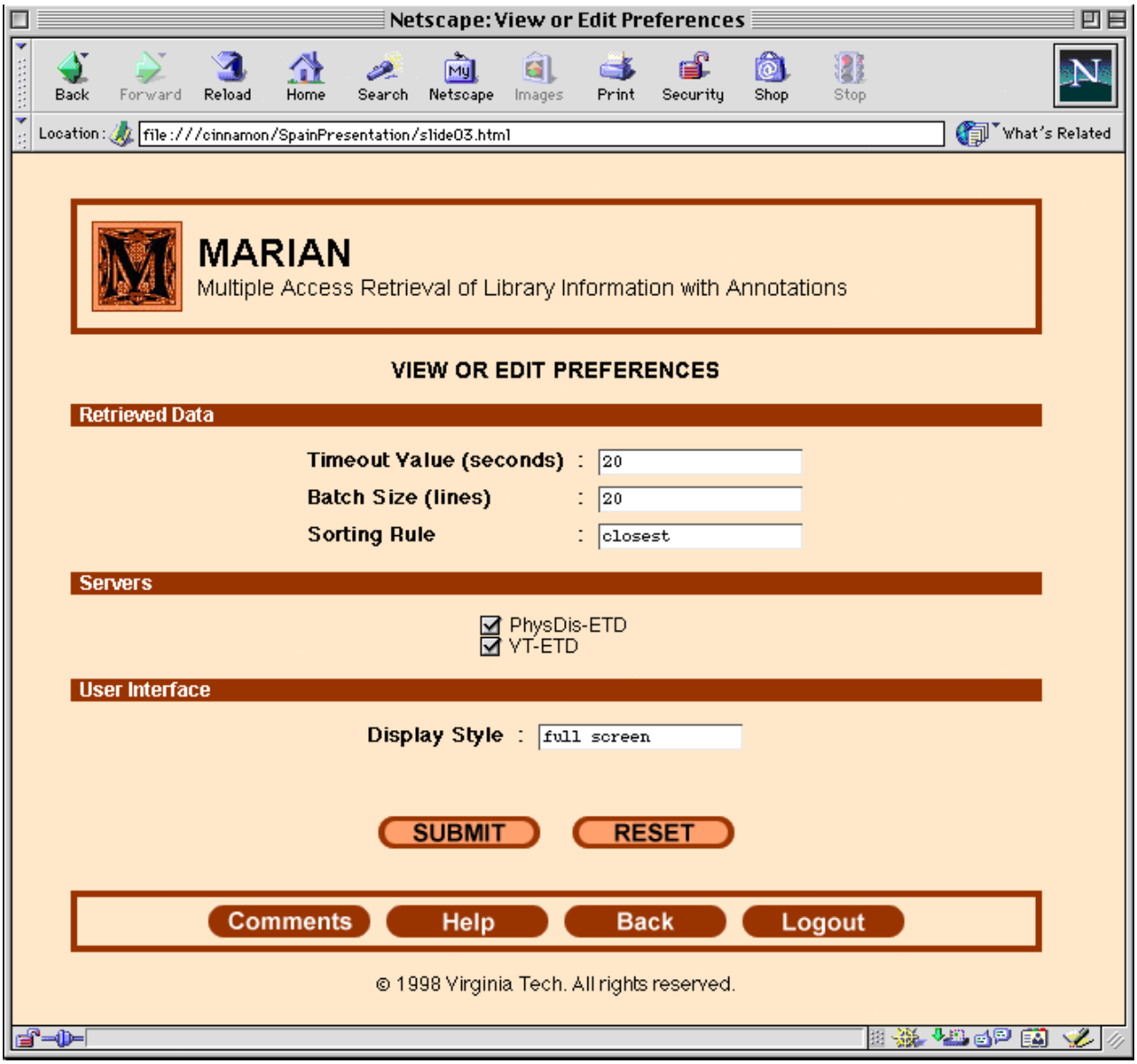


Figure 5.9. Default preferences in the prototype NDLTD union collection.

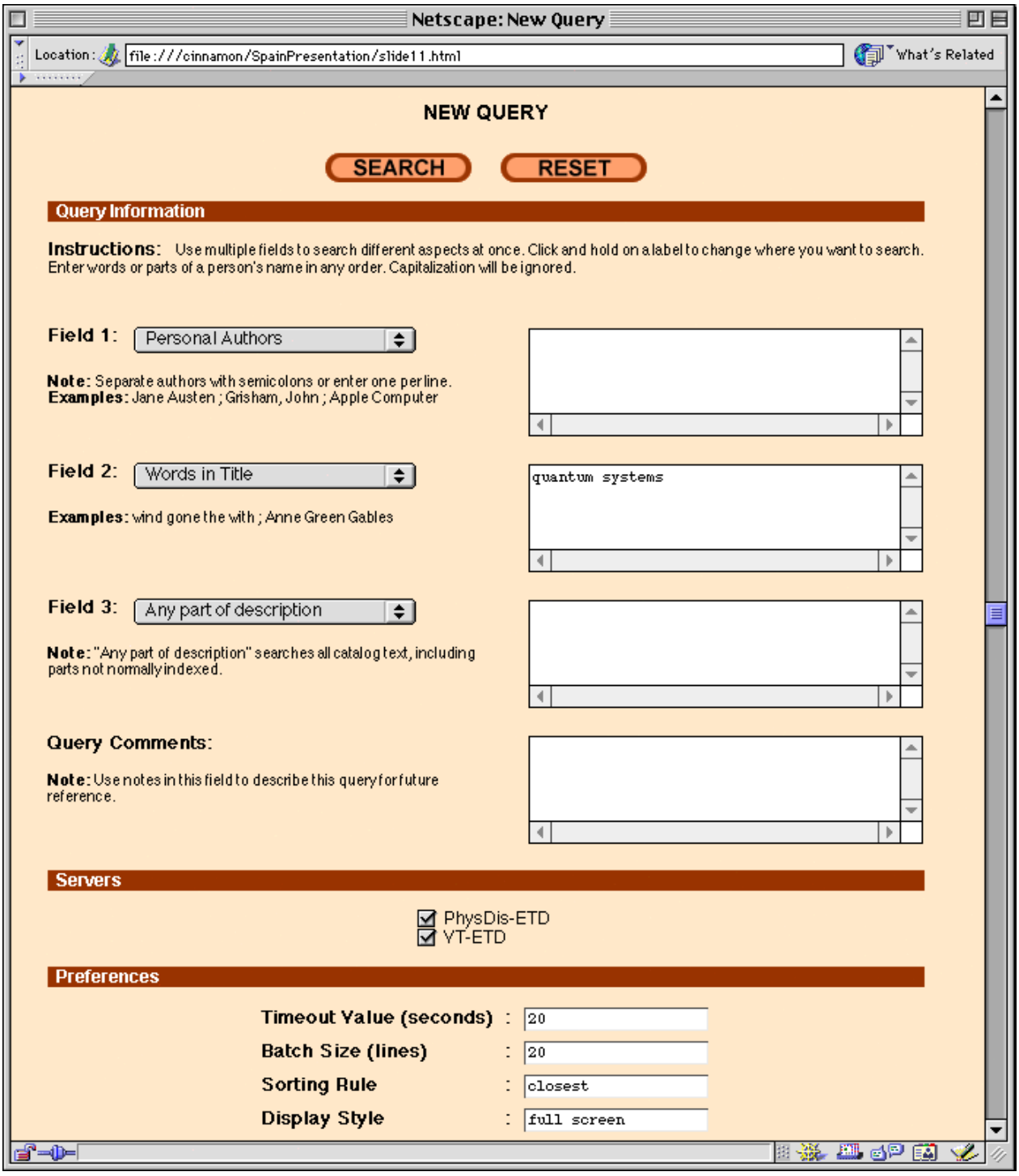


Figure 5.10. MARIAN query form for the NDLTD union collection.

Figure 5.11. MARIAN results page showing documents from both VT and PhysDis collections.

Figure 5.12. Full descriptions for the documents selected in Figure 10 shows the differences in format between VT and PhysDis collections.

5.5 References

Fox, E.A., et al, *Development of a Modern OPAC: From REVTOLC to MARIAN*. Proc. of the 16th International ACM SIGIR Conference on Research and Development in Information Retrieval, 1993: pp. 248-259

Fox, E.A., et al., *National Digital Library of Theses and Dissertations: A Scalable and Sustainable Approach to Unlock University Resources.* D-Lib Magazine, 1996. 2(8).

Fox, E.A., et al., *Networked Digital Library of Theses and Dissertations: An International Effort Unlocking University Resources.* D-Lib Magazine, 1997. 3(8).

Fox, E., *Networked Digital Library of Theses and Dissertations: An International Collaboration Promoting Scholarship.* ICSTI Forum, Quarterly Newsletter of the International Council for Scientific and Technical Information, 1997. 26: p. 8-9.

Fox, E.A., R. Hall, and N. Kipp, *NDLTD: Preparing the Next Generation of Scholars for the Information Age.* The New Review of Information Networking (NRIN), 1997. 3: p. 59-76.

Fox, E.A., *Networked Digital Library of Theses and Dissertations.* ERCIM News, 1998. 35.

Fox, E.A., G. McMillan, and J. Eaton. *The Evolving Genre of Electronic Theses and Dissertations*. in *Digital Documents Track of HICSS-32, Thirty-second Annual Hawaii International Conference on Systems Sciences (HICSS)*. 1999. Maui, HI.

Phanouriou, C., et al., *A Digital Library for Authors: Recent Progress of the Networked Digital Library of Theses and Dissertations*, in *Proceedings of the Fourth ACM Conference on Digital Libraries (DL '99)*. 1999, ACM: Berkeley, CA. p. 20-27.

France, R.K., L.T. Nowell, E.A. Fox, R.A. Saad, and J. Zhao, *Use and usability in a digital library search system*. CoRR cs.DL/9902013

Fox, E.A., *Networked Digital Library of Theses and Dissertations*, in *Proceedings DLW15*. 1999, ULIS: Japan.

Powell, J. and E. Fox, *Multilingual Federated Searching Across Heterogeneous Collections.* D-Lib Magazine, 1998. 4(8).

Hilf, E.A., *PhysDis, Physiks Theses around the World*, 2000. [http://elfikom.physik.uni-oldenburg.de/dissonline/PhysDis/]

Bowman, C.M., et al., *The Harvest information discovery and access system.* Computer Networks and ISDN Systems, 1995. 28(1): p. 119-126.

Van de Sompel, Herbert, Carl Lagoze. 2000. *The Santa Fe Convention of the Open Archives Initiative*. D-Lib Magazine 6, no. 2, [http://www.dlib.org/dlib/february00/vandesompel-oai/02vandesompel-oai.html].

OAI - Virginia Tech DLRL Projects. [http://www.dlib.vt.edu/projects/Oai].

Gonçalves, M.A., R.K. France, E.A. Fox, E.R. Hilf, M. Hohlfeld, K. Zimmermann, and T. Severiens, *MARIAN: Flexible Interoperability in a Federated Digital Library of Theses and Dissertations.* In Proc. 20th World Conf. on Open Learning and Distance Education, ICDE2001, Dusseldorf, GE, 1-5 April 2001.

Chapter 6: MARIAN: Flexible Interoperability for Federated Digital Libraries

6.0 Introduction

Networked or federated digital libraries are composed of autonomous, possibly heterogeneous information services, distributed across the Internet [14, 10]. The goal of federation is to provide users with a transparent, integrated view of such sources of information. Challenges faced include interoperability amongst different digital library systems/protocols [22], resource discovery (selection of the best sites to be searched) [12], and issues in data fusion (merging of results into a unique ranked list). In this paper we focus on the interoperability problem, one of the most challenging in the field of digital libraries. Heterogeneity occurs in both information representation and services, and must be addressed at four basic levels: system, structural, syntactic, and semantic [20].

One federated digital library where heterogeneity is a major problem is the Networked Digital Library of Theses and Dissertations [23], an international federation of universities, libraries, and other supporting institutions focused on efforts related to electronic theses and dissertations (ETDs). NDLTD has particular characteristics that complicate interoperability across member systems:

1. **Autonomy:** Members manage most services for their scholars.
2. **Decentralization:** Members are not (yet) asked to report either collection updates or changes in their metadata to central coordinators.
3. **Minimal interoperability:** Each source must provide unique URNs and metadata records for all stored works, but need not (yet) support the same standards or protocols.
4. **Heterogeneity:** Members differ in language, metadata, protocols, repository technologies, character coding, nature of data (structured, semi-structured and unstructured, multimedia), user characteristics, preferences, and capabilities.
5. **Massive amount of data and dynamism:** NDLTD already has over 120 members and eventually aims to support all those that will produce ETDs. New members are constantly added and there is a continuing flow of new data as theses and dissertations are submitted.

Due to the primary-source nature of ETD collections, the site selection process that is found in other systems (identifying a small number of candidate databases to search) is not always important here. For example, a query asking for new results in mathematics could retrieve information from almost every member university.

6.1 Federated Systems: Remote Search vs. Local Union

Transparent interoperability involves reconciling heterogeneity and integrating information sources at several levels [2]. A common architecture to deal with this problem uses mediators and wrappers [30]. Mediators export a common data model and provide a common query interface. Wrappers overcome some barriers of heterogeneity and produce source-specific queries. Wrappers also translate results between source and mediator data models. Within the mediated architecture there are two possible approaches to system integration [8]:

1. the union archive and
2. federated search.

In the union archive approach [26], information is periodically extracted from each source, processed, merged with information from other sources, and then loaded into a centralized data store – the union archive. Queries are posed against the local data without further interaction with the original sources. The main advantage of this approach is that adequate performance can be guaranteed at query time. On the other hand, union archives cannot guarantee delivery of the most current information to users. Concerns about data quality and consistency also must be addressed.

In the federated search solution, data remains at the sources and queries to the integrated system are decomposed at run time into queries to those sources. Data is not replicated and is guaranteed to be fresh at query time. On the other hand, more sophisticated query optimization and fusion techniques are required. Performance is also a drawback (see, e.g., [25]). Such factors must be considered as network latency and availability, amount of data to be transferred, etc. Overall performance is bounded by the worst-case situation.

In this paper we present MARIAN, an object-oriented digital library system, and demonstrate how we have used its modular architecture, flexible data model, and powerful search mechanism to create a federated system for NDLTD while addressing the problems described above. Due to poor and inconsistent network connectivities in the global NDLTD, variability in server load and administration, and the complexity of query translations in such a heterogeneous environment, we have chosen a union archive architecture for our integrated system. Components of our solution include: 1) the use and integration of several harvesting techniques; 2) a mediated union archive collection based on object-oriented ontologies of search modules and metadata; 3) a *collection view* mechanism for network representations comparable to database views; and 4) an integrated framework for addressing such questions as data quality, flexible and efficient search, and scalability. We use the unique characteristics of our system to build a common integrated solution for interoperability inside a unified framework.

6.2 The MARIAN Digital Library System

MARIAN is a search system for digital libraries [5, 7]. Originally designed for library catalogs, it has been used successfully for collections of varying sizes and structures, and has been enhanced to support digital library and semantic web [27] applications.

The MARIAN data model combines three powerful concepts. First, structure and relationships in MARIAN collections are captured in the form of an *information network* of explicit nodes and links. Similar graph-based models have proven effective in representing semi-structured data and Web documents [1], and for translating among different DL systems [17]. Second, MARIAN expands this model by insisting that the nodes and links of a collection graph be members of object-oriented *classes.* Classes are an organizing method similar to link labels in semi-structured graphs, but are strictly more powerful because they form a full lattice of subsets and can support inheritance. Furthermore, since nodes in the collection graph are instances of *information object* classes, they can support complex behaviors. In particular, they can support approximate matching of the sort pioneered in information retrieval (IR) systems. Third, nodes or links can be *weighted* to represent how well they suit some description or fulfill some role.

MARIAN is specialized for a universe where searching is distributed over a large graph of information objects. The output of a search operation is a *weighted set* of objects whose relationship to some external proposition is encoded in their (decreasing) weight within the set. Weights are used in IR, probabilistic reasoning systems, and fuzzy set theory. Our model grounds them firmly in a framework of weighted set operations [6] and extends them throughout the entire MARIAN system.

The use of object-oriented data and process abstractions in MARIAN helps to achieve physical and logical independence − common and useful concepts in the database field oft neglected in IR. Most current IR systems emphasize the physical level of term indexes and weight metrics, making it difficult to integrate systems at a conceptual level [11]. The flexibility of MARIAN's data model allows it to be used for object-oriented or semi-structured databases, knowledge representation, or IR. Its power comes from the smooth combination of a number of successful concepts from such fields and programming languages or artificial intelligence [9].

6.3 Harvesting Approaches

Any union archive approach includes: 1) mechanisms to gather or harvest data from the sources, and 2) some way of combining gathered data for use. This section covers harvesting approaches; Section 4 describes our architecture for combining harvested data.

Electronic theses and dissertations (ETDs) are large, sometimes archived in the form of several files. Many authors include multimedia material that would be difficult or impossible to include in printed publications. In response to this, a *de facto* standard has emerged at NDLTD sites of requiring a structured *title page* to serve both as directory to document files and as a convenient point for collecting and publishing metadata. Title page metadata are created by the author, usually with minimal oversight. At some sites additional metadata are added by trained catalogers. We choose to harvest all metadata – both controlled and uncontrolled – to create images of the sites in the union archive.

Much current work on federated DLs assumes a homogeneous structure or protocol (e.g., Dienst [14] or Z39.50 [16]) or a single means of harvesting (e.g., of HTML documents on the Web). In contrast, we work with several paradigms for harvesting data from heterogeneous sites, including

the paradigms of the Open Archives Initiative and Harvest™. In addition, a variety of data has been harvested using ad-hoc source-oriented approaches. The three approaches differ in the support they require from source archives.

The Open Archives Initiative (OAI) [15] is a multi-institutional project to address interoperability of archives and digital libraries by defining simple protocols for the exchange of metadata. The current OAI technical infrastructure is expressed by the Metadata Harvesting Protocol, which defines mechanisms for archives to expose and export their metadata. The OAI framework promotes an effective partial solution for interoperability, but particular archives must agree to implement the protocol and to export their metadata in a supported standard, which creates impedance to the solution. OAI emphasizes the distinction between data providers and service providers. The former manages a resource such as an e-print archive, acting on behalf of the authors who submit documents. The latter is a third party, creating end-user services based on data in archives.

The Harvest™ system [3] is a set of integrated tools for harvesting information from diverse repositories and building topic-specific content indexes. The architecture of the system is based on two main components: *gatherers* and *brokers*. Gatherers act as directed crawlers that collect and extract indexing and meta-information from repositories extracting summaries of content into a specific proprietary format (SOIF). Brokers provide the indexing and the query interface to the gathered information. They retrieve information from one or more Gatherers or other Brokers and incrementally update indexes. Although no metadata standard is enforced, external metadata standards (e.g., Dublin Core) can be incorporated.

We have faced situations where we cannot use any of these approaches, but where specific ways to gather data from sources exist. For example, in sources that use the Dienst protocol, specific combinations of services allow harvesting their data. The obvious drawback to ad hoc conversions is that they require development of specific solutions that are strongly dependent on the source.

6.4 The NDLTD Union Archive

In the prototype union collection described here, we have harvested metadata from four sources, each with its own formats. Table 6.1 summarizes the characteristics of each.

Collection	Harvesting Protocol	No. of records	Metadata format
PhysDis-ETD	Harvest	1256	SOIF – All DublinCore – 166
VT-ETD	OAI Z39.50	2427	ETD-MS – All MARC – All
MIT-ETD	Dienst	5000	RFC1807 – All

Table 6.1. Collections in the prototype union archive and their characteristics.

Just as there are many differences among institutions participating in NDLTD, there also are differences among the collections, especially regarding document format and access protocols. NDLTD did not specify standard formats or access protocols for documents or metadata. Although the adoption of standards is encouraged for NDLTD, it will be some time until a complete standardization takes place. Consequently our current union collection must cope with a multiplicity of formats, systems, protocols, etc.

Also, different collections support different document attributes and represent those attributes with different structures of data. Similar structures can be given different names by different sources, and structures with similar names may have very different semantics. For example, MARC records in the VT-ETD collection make a strong distinction between personal and corporate authors, while the *dc.creator* field of Dublin Core records may contain either. Again, some documents from the PhysDis collection are represented with Dublin Core metadata, including *dc.subject*, while others describe subjects with lists of automatically extracted keywords.

Thus, heterogeneity has several dimensions and induces four levels of interoperability concerns [20]: 1) system: which involves for example differences in harvesting protocols; 2) syntactic: including machine-dependable aspects of data representation; 3) structural: involving representational heterogeneity; and 4) semantic: with all the complexities related to meanings, significations, uncertainty, etc. In the following, we present how we use the unique characteristics of MARIAN to build an interoperability architecture that attacks each level of interoperability.

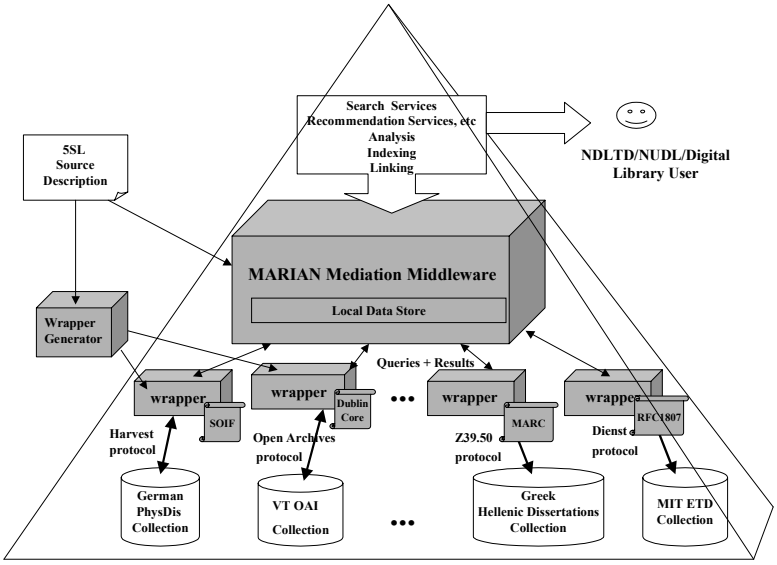


Fig. 6.1. The NDLTD Union Archive Architecture

6.4.1 MARIAN's Interoperability Architecture

The architecture of our system is presented in Fig. 6.1. The MARIAN Mediation Middleware provides the tools for structural and semantic interoperability. System and syntactic differences are addressed by wrapping sources with special software modules. Our 5SL language for declarative specification of digital libraries is used to describe capabilities of remote collections and their internal document structures. This information feeds data structures inside the mediator and allows semi-automatic generation of wrappers for harvested sources. Extended value-added services like searching, browsing, recommendation, personalization, and visualization are built on the top of the middleware.

6.4.2 System and Syntactic Interoperability

The harvesting process itself serves as a device to suppress some differences in source systems such as indexes and formatting, and helps towards systemic and syntactic homogenization. For example, textual information in different languages with different encodings can be locally homogenized to some standard like Unicode or UTF-8. Once we have harvested metadata from each remote collection and built local images for each, we can treat the local data with a unified set of text parsing, indexing and retrieval tools. Document (metadata) text fields such as *title, abstract,* or *body* are reduced to their individual terms using the same set of parsers, then matched to users' queries using the same search algorithms and ranking formula. This way we can ensure that the smallest atomic components, the text fields, will receive uniform treatment.

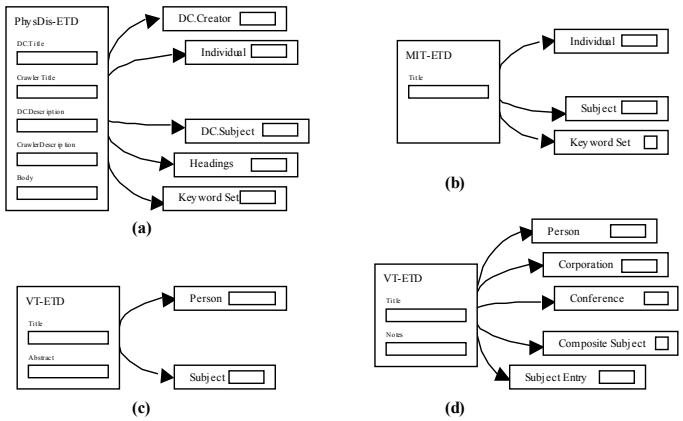


Figure 6.2. Images for (a) the SOIF PhysDis collection, (b) the RFC1807 MIT collection, (c) the ETD-MS VT collection, and (d) the MARC VT collection, all represented as class networks. Upward-curving links are (subclasses of) *HasAuthor* links; downward links, *HasSubject* links.

6.4.3 Structural Interoperability

Mediators map different representations of heterogeneous data sources to a common data model. Like many current approaches for data integration/mediation [4, 19], we use MARIAN's

network representation to overcome structural heterogeneity, capturing the structure of the remote collections as faithfully as possible. Figure 6.2 represents the document structures in each of our experimental collections.

In contrast to other network approaches, MARIAN's nodes and links are associated with object-oriented classes, which give us three major advantages. First, instead of using a single global searcher for the entire network, nodes and links are partitioned among class managers for a marked decrease in search complexity. Second, indexing and search are regarded as functional aspects of the classes, and thus can capitalize on regularities of the class. Third, the hierarchy of classes and search mechanisms provide a basis for the next phase of resolution of semantic interoperability.

6.4.4 Semantic Interoperability

Semantic heterogeneity is solved by exploiting two further MARIAN mechanisms: 1) semantically "tuned" but functionally equivalent searchers, and 2) a *collection view* ontology.

Nodes in the MARIAN information network can be simple atomic or scalar objects, as in the semi-structured model, but also they can be complex information objects. Information objects support methods proper to their classes, and all information objects in MARIAN support the method of approximate match to a query. For instance, MARIAN treats title text as a special sort of natural language sequence, with various rules for capitalization, punctuation, and sentence formation, but treats person's names as sequences of atomic strings. Matching methods vary from class to class but all have the same functional profile: given an object description of the appropriate type, they calculate how closely they match the description and return that value as a weight. Class managers draw on these methods to provide class-level search functions that, given an object description, return a weighted set of objects in the class that match the description. MARIAN already has stock matching functions and searchers for a number of common information object classes, a sample of which are shown in Fig. 6.3.

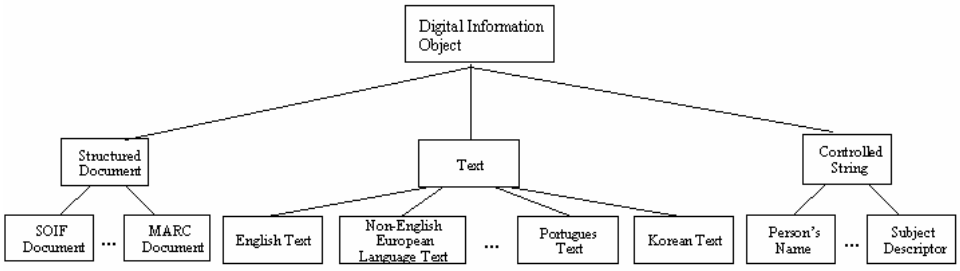


Figure 6.3. Part of the hierarchy of classes used in MARIAN.

Thus the first step in bringing a new document collection into semantic interoperability is to choose appropriate matching functions and searchers for the different objects in the collection.

Since class managers and searchers are object-oriented, specialized versions can often be easily created through inheritance. For truly different information objects new matching functions sometimes need to be defined, but even in this case stock searcher algorithms can often be reused. All that is necessary is to provide methods that follow the API of taking an object description to a weight or weighted set of objects.

Once a local image has been defined for an NDLTD member collection a view can be constructed. This involves defining a mapping to the member collection classes from a supported view. Such a mapping may use any combination of linking, inheritance and weighting. In the remainder of this paper we concentrate on one mechanism that has proven powerful and useful in NDLTD: the creation of synthetic weighted superclasses.

In NDLTD we are fortunate that a complementary interoperability effort [http://www.ndltd.org/standards/metadata/] has developed a metadata standard for electronic theses and dissertations (ETD-MS). Mapped into an information network model, this standard provides a stable view to the outside world for the union collection. A subset of the ETD-MS view is presented in Figure 6.4; to keep things simple we show only the attributes *title, creator, subject*, and *description*. The view ontology consists of three classes of objects, *ThesisDissertation*, *Individual* and *Subject*, together with *HasAuthor* and *HasSubject* links. The *Individual* class subsumes both persons and corporate individuals, while the *Subject* class covers diverse treatments. Mappings between the view and the underlying structures can be modified seamlessly.

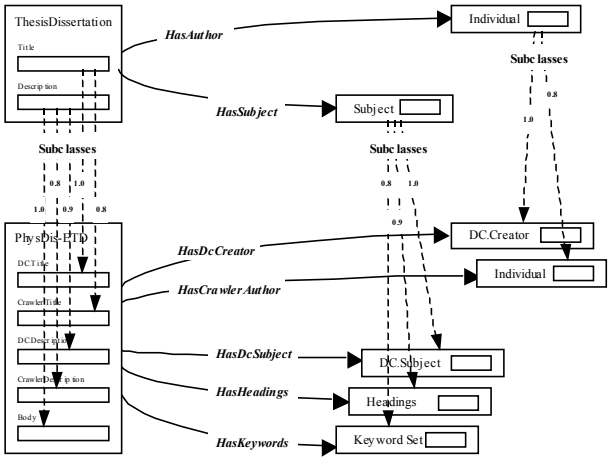


Figure 6.4. A collection view is abstracted from the PhysDis data to increase retrieval and usability.

In the case of the ETD-MS view of the PhysDis collection shown in Figure 6.4, all mappings make use of the weighted superclass construction. This construction asserts that all members of some specific class also are members of some view class, but that the extent to which they count as class members is different for different subclasses. In the case of PhysDis subject

descriptions, subclass relationships are weighted to reflect the authority of the description. In the next section we discuss the use of weights to address data quality issues. These uses interact, and the simple construct of synthetic superclasses with weighted subclasses cannot handle every situation, but we have found it strikingly effective.

6.4.5 Combining Heterogeneous Collections and Merging Ontologies

A direct approach for combining collection images into a union collection is depicted in Figure 6.5. This solution involves a fair amount of redundancy. For instance, several source images include classes for *Individual* and *Subject.* Such redundancy raises a design issue: what should be the ontology for the overall union collection?

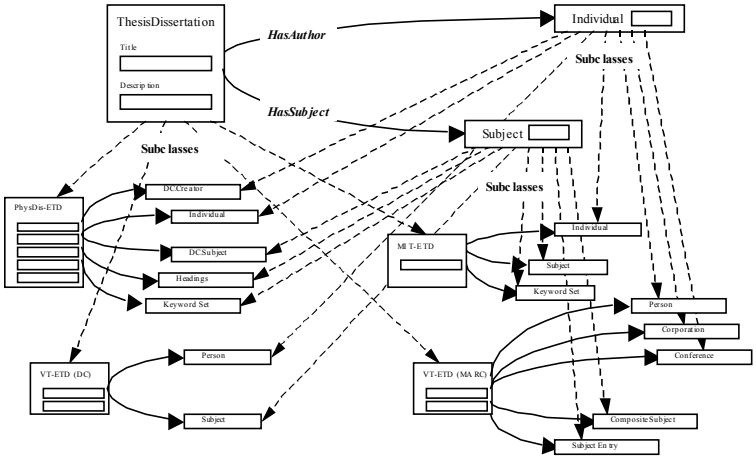


Figure 6.5. A direct approach for the union collection relates the views and images.

Figure 6.5 embodies one extreme where all the images are completely separate and only subclass-superclass relationships tie the object classes together. This approach has the disadvantage of data duplication: the same object (e.g., the subject heading "Computer Engineering") appears in several classes. Such redundancy wastes storage space in the class managers, and increases retrieval time when multiple classes are searched.

At the other extreme, we could immediately force all harvested data into our collection view by processing all incoming documents into structures with a single title and a single description field, all types of individuals into a single class, and all types of subjects and keyword lists into a class of subject strings. This would have the disadvantage of forcing us to combine fields as unlike as the PhysDis *Body* and *dc.description* fields into a single text, with corresponding losses to indexing specificity. It also would mean losing the information that sometimes we *do* know when an individual is a person, or when a subject heading comes from a controlled vocabulary.

Most importantly, however, pre-processing incoming data into the collection view ontology would mean giving up the ability to adjust to changing circumstances. Once our image of a

remote collection has been cooked, we can no longer reconstruct it in its raw state. On the other hand, the more original structure we retain, the better we can react to changes in the original collection, to addition of new collections, and especially to changes in semantic requirements.

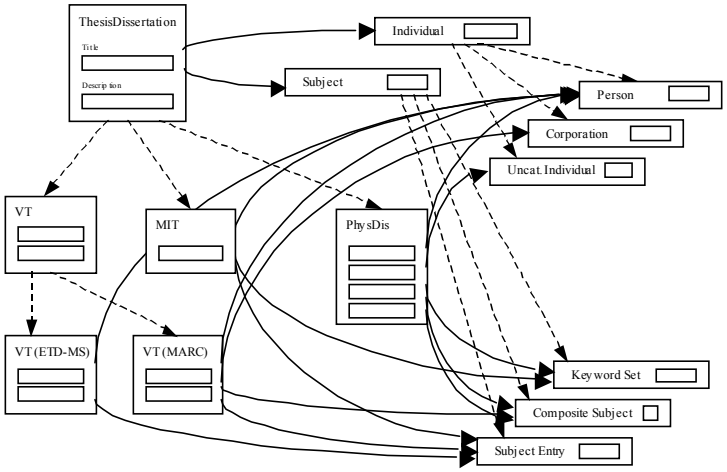


Figure 6.6. A more sensitive approach to the union catalog allows semantically similar object classes to overlap.

Between these two extremes lies a third alternative: merge image classes when these have sufficiently similar semantics, but keep classes separate when the semantics are different. Figure 6.6 shows this approach for the four images in the NDLTD union collection. The document hierarchy is as before. The *Individual* class has been analyzed into classes of (human) *Persons* and *Corporations.* SOIF, RFC1807 and Dublin Core author fields and MARC x00 fields, all of which require the name of a human person, are mapped to *HasAuthor* links to the *Person* class, while the MARC x10 fields produce links to the Corporation class. An *UncategorizedIndividual* class provides an image for those formats that make no such distinction, like the uncontrolled *author* field of the PhysDis collection. A similar breakdown of the *Subject* superclass into individual subject entries, composite strings with multiple entries, and sets of keywords provides image classes for all types of subject fields in the union collection. This approach simplifies the union collection ontology, with corresponding benefits in administration time and effort. It also saves string storage and retrieval time and adds functionality.

6.5 Solution Analysis

Combining weights, networks, and class structures enables us to both respect the data as it is harvested and provide simplified virtual collection views to users. It also makes it easy to change either the collection ontology or the underlying data without changing the view presented to the user, or to change the view presented to the user without restructuring the underlying representation or data. Moreover, it provides a unified framework to enhance retrieval

effectiveness in the union archive system by providing the flexibility to use different configurations and priorities on the same underlying data. In this section, we consider some properties we consider essential in any solution for interoperability.

6.5.1 Data Quality Issues

Data quality issues arise when one wants to correct anomalies occurring inside the integrated union archive to improve effectiveness of services. Examples of anomalies include errors in data, imprecision, multiplicity of representations, etc. In our architecture, we use MARIAN's weighting scheme as a way to mitigate data quality discrepancies.

The PhysDis collection provides a good example of the use of weights to enhance data quality. Each text class in the view corresponds to two or three classes in the underlying collection: a Dublin Core class and at least one uncontrolled class (Fig. 6.5). Our observations of the data indicate that the Dublin Core texts are of better quality than the uncontrolled texts. The superclass searchers capitalize on this by giving more weight to DC subclasses. In addition, the *Description* superclass depends more heavily on the PhysDis *Body* attribute than on either DC or uncontrolled description attributes, because we have observed that *Body* text tends to be a better representation of document content. All of these weights can be tuned with the increasing of experience with the union collection and with empirical experiments.

6.5.2 Efficiency

As seen, each MARIAN class manager functions as a searcher for objects in that class. All the searchers needed for the union archive are of five standard types: superclass searchers, text and structured document searchers, and weighted and absolute link searchers. The searchers are designed for optimal efficiency using three rubrics:

1. *Use all available information about the inputs*: searching can be a costly operation. Under certain circumstances, however, we can use information about the incoming sets to achieve better performance
2. *Capitalize on the power-law distribution*: Our observations of links in text and among digital library objects indicate that like small-world networks [29] they follow power-law distributions. Whenever possible, the MARIAN searchers are designed to run most efficiently when their inputs follow this distribution.
3. *Be lazy*: All searchers in the MARIAN community are designed to do only the work required to return as many elements as are requested. By design and construction, the first elements developed by any searcher are those with the highest weight. Lazy evaluation has its greatest pay-off in simple searches authored by human users, few of whom are interested in digesting more than a few dozen objects.

MARIAN searchers have been used for collections of up to a million objects and tens of millions of links, most noticeably in a "shadow" of the Virginia Tech academic library [7]. Response times for simple queries on collections of this size are comparable to other Net searchers, and remain acceptable for more complex queries. Research is currently under way to measure performance on collections of hundreds of millions of objects, and to verify the power-law model and its implications for searcher efficiency.

6.5.3 Scalability

Scalability, i.e., the ability to transparently and effectively grow a system, is a major concern in any platform based on integration of external sources. As important as large capacity are scalable query processing and the ability to incorporate new sources.

Just as analyzed objects are represented in MARIAN by graphs, queries are represented by relaxations of graphs and are processed following their structure. Elements of a MARIAN query are distributed to their governing class managers. Each class manager disassembles the portion of the query it receives; any parts it cannot handle are passed to others. Thus, an author / title search over the entire collection begins at the *Thesis-Dissertation* class. The title portion of the query is handled locally, but the author portion is passed to the link class manager for *HasAuthor* links, which passes the operation of finding matching people or corporations to the *Individual* class manager. Since query processing is distributed, it avoids evaluation bottlenecks and easily can be extended.

Easy incorporation of new sources is achieved by a mechanism for semi-automatically generating wrappers. Harvesting itself tremendously facilitates creating wrappers and communication with the mediator. In contrast to wrappers from other DL interoperation projects, which are *query processing oriented*, our wrappers are oriented towards *schema* and *ontological equivalence*. We have been using 5SL, a domain-specific language with a formal basis [13] designed for automatic generation of digital libraries, for describing external sources' capabilities for harvesting purposes. We describe external sources by their metadata structures and the correspondent harvesting protocols as scenarios (abstract sequences of events) that occur between the harvester/wrappers and the sources. Template wrappers can be configured (e.g., for periodicity of harvesting, additional mirrored repositories, or maximum number of harvesters acting at the same time). The wrappers currently in use in the union catalog, while originally crafted by hand, have since been successfully generated automatically from 5SL descriptions.

6.6 References

[1] Abiteboul, S., Buneman, P. Suciu, D., *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 1999

[2] Adam, N., Atluri, V., Adiwijaya, I., "Systems Integration in Digital Libraries", *Communications of the ACM*, **43**(6), 2000, pp. 64-72

[3] Bowman, C. M., Danzig, P. B., Hardy, D. R., Manber, U., Schwartz, M. F., "The Harvest information discovery and access system", *Computer Networks and ISDN Systems*, **28**(1–2), 1995, pp. 119–126

[4] Fernandez, M. F., Florescu, D., Levy, A. Y., Suciu, D. "Declarative Specification of Web Sites with Strudel". *VLDB Journal* 9(1): 38-55 (2000)

[5] Fox, E.A., R.K. France, E. Sahle, A.M. Daoud, and B.E. Cline, "Development of a Modern OPAC: From REVTOLC to MARIAN". *Proc. 16th Int. ACM SIGIR Conf.*, 1993: pp. 248-259

[6]France, R.K. "Weights and Measures: an Axiomatic Approach to Similarity Computations". Internal report, Virginia Tech, 1995; http://www.dlib.vt.edu/repors/WeightsMeasures.pdf

[7] France, R.K., L.T. Nowell, E.A. Fox, R.A. Saad, and J. Zhao: "Use and usability in a digital library search system." CoRR cs.DL/9902013:

[8] Florescu, D., Levy, A., Mendelzon, A. "Database techniques for the World-Wide Web: A Survey", *SIGMOD Record*. **27**(3) 1998, pp. 59-74

[9] Fuhr, N., Rolleke, T., "A Probabilistic Relational Algebra for the Integration of Information retrieval and Database Systems", ACM Transactions on Information Systems, Vol. 15, No. 1, January, 1997, Pg. 32--66.

[10] Fuhr, N. "A Decision-Theoretic Approach to Database Selection in Networked IR". *ACM Transactions on Information Systems* 17(3): 229-249 (1999)

[11] Fuhr, N., "Towards Data Abstraction in Networked Information Retrieval Systems", *Information Processing and Management* 35(2): 101-119 (1999)

[12] Gravano, L., Garcia-Molina, H, "Merging Ranks from Heterogeneous Internet Sources", *Proc. of the 23rd International Conference on Very Large Databases, 1997*, pp. 196-205

[13] Gonçalves, M.A., Kipp, N.A., Fox, E.A., Watson, L.T., "Streams, Structures, Spaces, Scenarios and Societies (5S): A Formal Model for Digital Libraries", Tech. Rep., Virginia Tech, 2001.

[14]Lagoze, C., Fielding, D., Payette, S., "Making Digital Libraries Work: Collection, Services, Connectivity Regions, and Collection Views", *Proc. 3rd ACM Digital Libraries*.1998, pp.134-143

[15] Lagoze. C., Sompel, H. V., "The Open Archives Initiative", Proc. of the First ACM-IEEE The Joint Conference on Digital Libraries, Roanoke, Virginia, 2001.

[16] Lynch, C., "The Z39.50 Information Retrieval Standard - Part I: A Strategic View of Its Past, Present and Future", *D-Lib Magazine*, April 1997.

[17] Melnik, S., H. Garcia-Molina and A. Paepcke, "A Mediation infrastructure for digital library services" *Proc. 5th ACM Digital Libraries, San Antonio, 2000* pp.123-132.

[19] McBrien, P., Poulovassilis, A., "Automatic Migration and Wrapping of Database Applications - A Schema Transformation Approach". ER 1999: 96-113

[20] Ouksel, A. M., Sheth, A. P., "Semantic Interoperability in Global Information Systems: A Brief Introduction to the Research Area" *SIGMOD Record* 28(1):5-12 1999

[22] Paepcke, A., Chang, C. K., Winograd, T., Garcia-Molina, H., "Interoperability for digital libraries worldwide." *Communications of the ACM* **41**(4), 1998, pp. 33–42.

[23] Phanouriou, C., Kipp, N. A., Sornil, O., Mather, P., Fox, E. A., "A Digital Library for Authors: Recent Progress of the NDLTD", *Proc. 4th ACM Digital Libraries*, 1999, pp. 20-27

[25] Powell, A.L. and J.C. French, "Growth and server availability of the NCSTRL digital library." *Proc. 5th ACM Conf. On Digital Libraries (San Antonio, June 2-7, 2000)* pp. 264-265.
[26] Rundensteiner, E., Koeller, A., and Zhang, X., "Maintaining Data Warehouses over Changing Information Sources", *Communications of the ACM*, **43**(6), 2000, pp. 57-62
[27] Semantic Web Activity; http://www.w3.org/2001/sw/
[29] Watts,D. J.,"*Small Worlds:The Dynamics of Networks between Order and Randomness",* Princeton Univ. Press, 1999.
[30] Wiederhold,G.,"Mediators in the Architecture of Future Information Systems", *IEEE Computer*, **25**(3),1992, pg. 38-49.

Curriculum Vitæ
Robert Karl France

Home:
2502 Carolina Ave. SW
Roanoke, VA 24014

Work:
Digital Library Research Laboratory
2030 Torgersen Hall
Virginia Tech 0368
Blacksburg, VA 24061

540.231.6256

france@vt.edu
http://opac3.cc.vt.edu/france/

Education:

Sept. 1999 - present	Virginia Tech, Blacksburg VA.	**PhD in Computer Science and Applications. Thesis Title:** Effective, Efficient Retrieval in a Network of Information Objects **Preliminary Exam:** 18 January 2001. **Expected Completion Date:** August 2001.
Aug. 1984 - July 1986	Virginia Tech, Blacksburg VA.	**MS in Computer Science and Applications. Areas of concentration:** information retrieval, artificial intelligence, theory of computer science, programming languages. **GPA:** 3.9. **Honors:** inducted Sigma Xi, given rare Graduate Project Assistantship, three times awarded non-service Instructional Fee Scholarship.
Jan 1984 - May 1984	University of North Carolina, Chapel Hill, NC.	Full-time graduate student in computer science. **Areas of concentration:** software engineering, theoretical computer science. **Average:** H.
Aug. 1980 - Dec. 1983		Part-time graduate student in computer science, evening college. **Average:** P.
Aug. 1975 - May 1978		Graduate student in philosophy. **Areas of concentration:** ethics, logic, history of philosophy. **Average:** H-.
Sept. 1970 - Dec. 1973	Oberlin College, Oberlin, OH.	**BA in philosophy**, minor in mathematics; coursework in computer science, including business programming and artificial intelligence. **GPA:** 3.4.

Employment:

Teaching:

Dec 1996 - June 1998	Community School, Roanoke, VA.	**Subject / Specialist Teacher**: Elementary school computer concepts and operation.
Sept 1986 - June 1987	Hollins College, Roanoke, VA.	**Instructor**: Computer Concepts through Word Processing, Pascal.
Sept 1986 - Dec 1986	Virginia Tech, Blacksburg, VA	**Instructor** (at rank of GTA): Data Structures.
Jan 1984 - May 1984	Department of Computer Science, University of North Carolina, Chapel Hill, NC.	**Teaching assistant**: Data Structures.
Aug 1975 - May 1977	Department of Philosophy, University of North Carolina, Chapel Hill, NC.	**Instructor** (at rank of TA): Symbolic Logic. **Teaching assistant**: Introduction to Philosophy, Ethics, Symbolic Logic.
Feb 1973 - June 1973	Oberlin College, Oberlin, OH.	**Teaching team member** (math and computer science): Form in Nature.

Computer Systems Design and Implementation:

Aug. 1998 - present	Digital Library Research Laboratory, Virginia Tech, Blacksburg, VA.	**Researcher.** Initiation and oversight of new and continuing Digital Library research. Project coordination and supervision, including evolution and maintenance of MARIAN system, re-engineering of MARIAN in Java, and TREC participation. Responsible for Laboratory environment and interaction with other University departments. Laboratory Major Domo and Webmother. **Supervisor:** Edward A. Fox.
Jan 1990 - July 1998	Computing Center, Virginia Tech, Blacksburg, VA.	**Programmer/Analyst.** Design and implementation of advanced information systems. Oversaw and participated in the implementation of the MARIAN library retrieval system, including production of new code, improvement of experimental software to production quality, data processing, and system maintenance. Design and production team member, Envision digital library system. **Supervisor:** Edward A. Fox.

Aug. 1987 - Dec. 1989	Virginia Tech, Blacksburg, VA (NSF and other grants).	**Research associate.** Design, data preparation and implementation for very large natural language word knowledge base. Creation of knowledge representation (KR) language and communication system, support system for distributed, concurrent programming, and KR-oriented database system. Expert systems and interface programming. **Supervisors:** Edward A. Fox and J. Terry Nutter.
Dec. 1984 - Dec. 1985	Department of Computer Science, Virginia Tech, Blacksburg, VA.	**Research assistant.** Systems analysis, design and programming on CODER expert information retrieval system. **Supervisor:** Edward A. Fox.
June 1972 - Aug. 1972	Information Associates, Inc., Rochester, NY	**Programmer.** Wrote and maintained commercial program segments in FORTRAN and COBOL. **Supervisor:** Gwyneth Jones.

Library Work:

Aug. 1978 - Aug. 1983	Academic Affairs Library, University of North Carolina, Chapel Hill, NC.	**Microforms Collection Supervisor.** Maintained and developed a collection of over a million units, a public-service area including circulation, reference and interlibrary loans, and a copy and duplication service. Responsible for physical re-organization of the collection, new cataloging and access techniques; statistical, clerical, and administrative procedures, and planning for new library building. **Supervisor:** Carson Holloway.

Service:

Community School Board of Trustees: member, 1992 - 2000; chair, Long-Range Planning Committee, 1996 - 2000; chair, Building Committee, 1998 - 2000.

Nature Conservancy: Bottom Creek Gorge monitor, 1993 - present.

Roanoke Valley Preservation Foundation: President, 1994. Board member, 1991 - 1997; 2001 - present.

New Century Council: member, Technology committee, 1994 - 1995.

Dead Logicians Society: member, 1990 - 1996.

> The Dead Logicians are an intercameral group of mathematicians, philosophers, and computer scientists from Virginia Tech, Roanoke College and Hollins College who meet regularly to discuss mathematical and formal systems of mutual interest.

Virginia Tech University Library Committee: graduate student representative, 1985 - 1986.

Joint Conference on Digital Libraries 2001: local arrangements chair.

Professional Associations:

Association for Computing Machinery

Institute for Electrical and Electronic Engineers, Computer Society

American Society for Information Science and Technology

Association for Computational Linguistics

Sigma Xi
