The Academy:  
A Family of Search Engines for Digital Libraries

Robert France  
Draft 1.1  
6 September 1999

Introduction and Motivation

0.1 The Document Universe

[[ It is very large. It is dynamic. ]]  
[[ It is not just made up of documents. ]]  

Many sorts of objects inhabit the digital library. Pride of place, of course, go to digital information objects themselves, as these are the reason for the library’s existence. Information objects run the gamut of types, sizes, and formats from small packets of ASCII text, through sound and video recordings of various lengths up to huge sets of scientific sensor data. They have only three things in common, captured by the three words digitalinformationobject: they can all be represented by streams of bits, they all purport to carry information of interest to human beings, and they are all unique individuals of their type.

Second in importance only to the information objects in a digital library are their metadata surrogates. These shadowy entities are information objects in their own right (authored as a rule by catalogers, although other sources are possible) but very specialized ones. Metadata objects generally have a fixed type and a common size, and carry very specific kinds of information to human beings or computational systems: information that describe the library documents in ways that directly support their classification, maintenance, and retrieval. Typical metadata elements include authorship, provenance, version information (including various important dates) and requirements for presentation.

In a traditional library, only the metadata are the proper domain of searching tools such as card catalogs and OPACs. The digital library blurs this distinction. In the digital library, both metadata records and repository documents are digital objects stored in and accessed through library structures. The distinction of use is still very clear, however. A metadata object is created from an existing object with the purpose of providing descriptive information tailored to one or more classes of library users. A metadata record is always created as a surrogate for something other work.

Information objects are not the only sort of entities with surrogates in the digital library. Other important classes of entities that participate in the universe of information include people — as authors, editors, illustrators — institutions — as authors, sponsors, work places for people — and controlled concept systems — flat, hierarchical, and network. All of these classes of non-digital
objects have their digital surrogates in the library. People are shadowed by records in authority files, which capture details such as their names, work places, and the important dates in their lives. Authority records attempt to catalog and control variant forms under which a given person has published with the goal of uniquely determining individual authors. Other structures track organizations and classification systems. In traditional libraries these tools have tended to be physically separated — often in different departments. They have come in a bewildering array of different formats and media, and only some have been available to the public. But they have been part of the structure and content of libraries for centuries.

One other sort of (radically non-digital) object to have surrogates in a library is the abstract work. *Romeo and Juliet* is an abstract work. Each version of the play from the first folio to recent movie scripts serves in some sense as an instantiation of the work, although each version is also clearly a unique information object in its own right. The exact relationship between abstract and concrete works is notoriously difficult — it is, for instance, one of the muddier aspects of the MARC metadata representation system — but libraries can scarcely do without some system of abstract works, if only to represent the subjects of studies on *Romeo and Juliet*.

Finally, the digital library includes relationships. There are relationships between information objects: this scholarly paper is a the next version of that one; this set of regulations supersedes that one; this paper comments on that one. There are relationships between concrete and abstract works: this is an edition of that; this is a translation of that. The relation of authorship ties works (both abstract and concrete) to people, organizations, and (in a less common but meaningful sense of authorship) conferences. Cataloging systems teem with relationships, as do lexicons and encyclopedias. All of these are necessarily a part of the document universe, and of the digital library.

### 0.2 Text, Structured Text and Hypertext

[[ Not all documents in a digital library are made up of text, but text objects and text fields associated with non-textual objects remain an important area, particularly for searching. ]]

We commonly picture text as a sequence of words; or alternatively as a sequence of paragraphs, each of which is composed of a sequence of sentences, each of which is itself a sequence of words. It is worth noting, however, that even in the most impoverished text production systems, this has never been completely true. Formatting information, if only in the form of explicit character returns and extra spaces, has always been used and been significant. [[...]] With the advent of mark-up languages and with the proliferation of word processors and desk-top publishing software, document structure is becoming more common. Fortunately, it is also becoming more standardized, particularly with the use of languages like SGML and XML. As structure becomes more accepted in the document universe, users become both more ready to use it is queries and less tolerant of systems that ignore it. [[...]]

It is also worth noting that text is not so much a sequence of *words* as a sequence of *terms*, including most commonly words, but also including names, numbers, code sequences, and a variety of other $*$&@ tokens.

Just as we commonly simplify text into a sequence of words, so too it is common in information retrieval to regard documents as single texts. Nothing is less common, however, than a document with only a single part and that unstructured text. Typical documents have titles, authors, abstracts, distribution lists, synopses, subject lines, lists of keywords, and/or any of a host of dates.
The natural extension of this concept is to the universe of hypertext, where texts, structured texts, and compound documents are linked together by relationships including ... Search and retrieval in such a universe involves new questions: Where does a document begin and end? How can we decide how much to show to a user? When does a query need to be matched by a single node in a hypertext, and when may partial matches in several nodes count?

0.3 Approximate Matching

One way in which information retrieval systems tend to differ from classical database systems is that the criteria for match of a text or a document to a query tend to be approximate. ... Ranked retrieval ... Boolean systems & extended Boolean systems

1. Foundations

1.1. Links and Objects

MARIAN and the Academy are grounded in a universe made up of objects and links. This is a powerful and flexible representation scheme that has proven useful in semantic networks, computational lexicography [], and hypermedia [CHEN, FOX & FRANCE][THE OTHER CHENN, FOX, & FRANCE]. It is also a form of knowledge representation that occurs naturally in digital library ontologies. Authority files and subject classification schemes are both best understood as collections of objects (names or the one hand and descriptors on the other) together with well-defined categories of links. The links within the Library of Congress Subject Headings are a particularly interesting and rich set, including see, see also, and XXX links along with links connecting superseded categories to newer ontologies and the hypernymy / hyponymy links that establish LCSH ontological hierarchy.

The wider bibliographic universe described in library metadata is also well captured by an object-link representation scheme. Works in the (digital) library are embodiments of abstract works authored by people, corporations and conferences, any of which or whom may occur as subjects of other works, along with abstract subject descriptors ... the list could continue. Suffice it to say that this is both a natural way of speaking about the contents of digital libraries – both their indexing structures and their underlying collections – and a powerful one.

There are a number of ways to formalize this representation. Perhaps the most natural is to look at it as a specialization of the entity-relationship model [], where entities are generalized to typed objects in the object-oriented sense, and relations are simultaneously restricted to directed binary relations. Alternatively, we could look at it following the SNEPS model [] as a form of propositional logic. We will take a mathematically minimalist approach based on sets and ordered pairs.

If we posit a universe of objects, then we can consider types as subsets of the universe. These types form a lattice that is a subspace of the power set of objects: that is, each type is a subset of the set of all objects, and for each two types there is both a least common supertype and a greatest common subtype. Sometimes the least common supertype (the smallest type that includes all objects of either type) is the universal type object and sometimes the greatest common subtype (the
largest type that is included in both types) is the empty or absurd type \textit{null}. Typically these
developed types, often called meets and joins, will be unique. Certain models involving infinities
of objects lose this useful property, but those too can fall within our model.

The primary purpose of types in the model is to specify the domains and ranges of functions. Each
type of object in a universe is associated with certain particular functions. In the object-oriented
model, we would call these the methods of the type, which we would then refer to as a class. Of
particular interest in the digital library domain is a class of matching functions that determine how
well an information object matches a user’s information need. These functions may behave very
differently from type to type of object: an information description might match a title very well and
a subject descriptor not well at all even if the text value of the title and descriptor are identical. This
is because the text value of a subject descriptor is not its most important aspect: far more important
is its position in an ontological hierarchy.

From the construct of a set, we develop the concept of an ordered pair in the usual way. An
ordered pair is an indexed set of two elements, each drawn from the universal set of objects. If the
universe contains \( N \) elements, there are \( N \cdot N \) elements in the set of all possible ordered pairs, and
\( 2^{N \cdot N} \) possible sets of ordered pairs. Each such set can be regarded as a link class. A set of
ordered pairs is called a relation; a set where each ordered pair has a unique first item is called a
function. Relations form a type lattice within the power set of all ordered pairs. Thus each link
class can be regarded as a set relation, and for each two link classes we can find or generate a meet
and a join.*

_____________________

* Purists will note here that in the case of both links and objects we are conflating \textit{intension} and
\textit{extension}; that is, we are asserting that the semantic meaning of a type and the exact set of objects
that it includes are one and the same. This is a move of dubious philosophical pedigree, since it
seems that we can give meaningfully different semantic descriptions of a single set of items. It has
more validity in mathematics, however, where in a well constructed model two sets will have the
same extension in exactly the case where they have equivalent semantic descriptions. This is one
of the bases for mathematical model theory.

_____________________

1.2. Weighted Objects and Weighted Links

When we come to performing retrieval operations on object-link structures, it will help to add one
further enrichment to our model. Retrieval brings us the idea of an approximate match to a
statement of information need. We choose to represent the strength of such a match by a \textit{weight}: a
quantity that runs continuously from \textit{not at all} to \textit{perfect}. These two limits, which we will refer to
as \textit{bottomWt} and \textit{topWt} respectively, mean that weights form a closed set. They must also form an
ordered set: we will require that for any weights \( w_1 \) and \( w_2 \)
\begin{itemize}
  \item either \( w_1 \leq w_2 \) or \( w_2 \leq w_1 \)
  \item both \( w_1 \leq w_2 \) and \( w_2 \leq w_1 \) when and only when \( w_1 = w_2 \).
\end{itemize}
The simplest model for weights is thus a closed interval on the real number line. Without loss of
generality, we will use the interval \( 0..1 \).*

_____________________

* It may be pointed out that this is not the only reasonable model for weights. The most important
objection is that it brings in much more machinery than is called for by the few axioms above. In
particular, once one starts thinking of weights as real numbers, one is immediately tempted to add
and multiply them ... and to use bottomWt and topWt as additive and multiplicative identities as
well. In many approximate retrieval systems this temptation is justified, but not in all.

Once we have introduced weights we can extend the model to associate with every object a [ range
of ] weighted objects and with every link a [ range of ] weighted links (Fig. WEIGHTED_MODEL). This association effectively replaces each object in our universe with an
ordered pair and each link with an ordered triple. All other properties of the universe remain
unchanged. How we interpret such a model will depend on the weighting scheme imposed. In
particular, different schemes will require different constraints on how objects and links can be
assigned weights (in particular, how linked objects or sets of links with a common endpoint may
share weight) and different rules for propagating changes of weights through the network. We
will describe how this works later.

In one sense, we have now replaced each model of the domain of information objects by
uncountably many models, each with different weight values for each object and link in the
universe. And it is true that the same collection of objects and links can create a dense shadow in
the weighted object-link model. This is more or less inevitable if we are to model digital library
spaces, since different retrieval systems can produce very different similarity values from the same
data [RORVIG99]. This effect should not concern us on either a practical or a theoretical level,
however. On the theoretical level, we note that we are not in effect using the full power of the real
numbers for weights, and that the axioms we do require pertain only to comparison. On the
practical side, we note that precise weights are notoriously hard to derive in information systems,
and that in fact a finite range of integers (say, from 0 to $2^{16}-1$) is sufficient for generating and
manipulating the weights that actually arise in information systems.

1.3. Weighted Object Sets

[[ Two equivalent views: Woset as sets of weighted objects and Woset as ordered pairs,
set+weighting function. ]]

That is the extent of the representational machinery needed for this model. Its limitations are
obvious and many. On the other hand, I hope to establish that this model carries a great deal of
power – power that is particularly appropriate in the related domains of information systems and
digital libraries. It has the advantage of simplicity. And it has the incomparable advantage that in
finite or countably infinite domains it is decidable [MODEL THEORY].

2. Application

2.1. Objects, Classes, and Class Systems

The digital library domain is a domain of digital information objects — texts, images, videos, and
so forth. As digital libraries emerge from the research community into full use, the “and so forth”
comes to include more and more categories of objects, including objects with complex parts, sets
or collections of objects, and surrogates for real-world entities such as authority file records, which
stand as surrogates for persons and corporate individuals in the real world.
Faced with such a profusion of types, is is only prudent to use as general a model as possible. Accordingly, we will begin by considering our domain to be simply objects of any type. We ask three things of our objects, two as part of the abstract model and one as an implementation tool. In the abstract we ask of objects that they have type and identity. Specifically, any object in our domain must fall into a particular object class in some determinable class lattice (usually a simple fixed class hierarchy, but possibly a dynamic type system with multiple inheritance). It is not necessary for the type of an object to be directly observable — text objects like “mark twain” may have a type determined only by the context in which they are found — but each object must have some type.

One primary function for types is to set the criteria for object identity. Our model requires that object identity be determinable. For any two objects in our universe, there must be a computable test that we can perform to determine whether they are or are not the same object. That test is a function of the objects’ common class. For any two objects in the same class, we expect the class to define the identity test. In the digital library domain, for instance, the classes works, free text terms, and persons all have very different identity criteria, none of which are simple comparison operators of the sort used for programming language classes like integers or character strings.

When objects are of different classes, we must resort to the class hierarchy. When comparing an object of one class with an object of an ancestor class, we will use the ancestor class criteria. When comparing objects of sibling classes, we will use the parent class criteria. In general, when comparing objects of any two commensurate classes, we will use the criteria of the most specific common superclass. Objects of incommensurate classes (in the case where the class structure is not a single tree or lattice but a proper forest) would by nature be non-identical. We expect any domain of digital information objects, however, to have a singly rooted class structure, where we will refer to the root class as object. It is not too much of a strain to envision the identity criteria for object: it can only be direct bitwise comparison of the two objects. Any objects that are bitwise identical must be the the same in any digital information object class. Criteria for more specific classes can only become more lenient as the class acquires more semantic constraints.

Finally, we require that the classes in our domain have implementations that include the ability to determine object identity and to map each unique object into a unique instance ID. An object is thus completely described by a class ID plus an instance ID, or what we term a full ID. In MARIAN and related Virginia Tech systems, a class ID is implemented as a 16-bit unsigned integer; and an instance ID as an unsigned 32-bit integer. Other implementations can easily be imagined, but this one has proven effective and resilient. Most object classes have associated class managers: single computational objects residing at fixed locations on the Net that handle translation between objects and IDs. In the case of digital information objects, class managers also include databases and files for object storage.

Once we have a universe of objects, we can obtain an arbitrary amount of structure over that universe using classical set theory. The universe of digital information objects does not require the full power of set theory. For one thing, we can be assured that there are at most a countable number of objects in the universe, since they must by their nature be representable using bit streams. (In point of fact, any actual digital library will be not only countable but finite. Our abstract model will not suffer, however, if we allow it to include countably infinite sets of objects and objects represented by infinite bit streams. In fact, it will make the model simpler to allow these. Furthermore, while there are no objects or sets in the digital library domain that are actually infinite, neither is there any upper bound to the size of objects or sets that can be part of a digital library. So, as usual in mathematical modeling, we will read “arbitrary size” as “infinite size,” which keeps the argument simple.)
2.1. Weighted Objects and Weighted Object Sets

We will also find it useful to consider weighted objects and weighted object sets. A weighted object is just an object with an associated weight; a weighted object set is a set where every member has an associated weight. In the implementation used for the Academy, a weighted object is always in actuality a weighted object surrogate, consisting of a class ID, an instance ID, and a weight. Weighted object sets are implemented in different ways depending upon the type of set, but generally also make use of full IDs rather than full digital objects. It may be pointed out that from an abstract point of view a weighted object (separate from a weighted object set) is decidedly a second class entity. In implementation, however, it proves useful to be able to discuss and manipulate individual weighted objects, ignoring temporarily where their weights come from.

Before we can discuss weighted objects and weighted object sets we must define what we mean by a weight. A weight is a comparable, scalable, additive quantity that runs from some lower limit bottomWt to some upper limit topWt. It is used to represent quantities like closeness of fit, strength of belief or confidence, all of which have the characteristic of being limited on both ends. Closeness of fit runs from “completely dissimilar” to “exactly the same;” belief runs from complete disbelief to complete certainty; confidence runs from none to total. For simplicity, we will model weights as continuously varying quantities, although in practice our precision in judging weights is low enough that we can implement them with 16-bit unsigned integers. How weights combine depends on their semantics. An analysis of the algebra of weights derived from matching functions can be found in ["Weights and measures"]. For the purposes of our model, however, we need only recognize that weights can be combined, and that the category Weight is closed under the combining operations.

If a weighted object is an object with an associated weight, a weighted object set could be defined as a set of weighted objects. It is more revealing, however, to consider a weighted object set as a set of objects together with a weighting function that maps each object in the set into a unique weight. Thus the category Woset is a category of ordered pairs where the first member is an element of Set and the second an element of the category of all functions from Object to Weight.
// defining only constructors, destructor, and these two methods.
// Most child classes actually re-define some subset for the
// purposes of efficiency, but this implementation provides a
// rock bottom.

class wtdObjSet
{
public:
    virtual ~wtdObjSet() {}; // Should be = 0, but pure virtual
    // destructors are giving G++ problems now. RKF 6-17-93

    virtual Pix first();
    virtual Pix skipToKth(int K);
    virtual void next(Pix& p); // Advance p to next object.

    virtual wtdObj operator()(Pix p) = 0;
    virtual int sample(int Num, wtdObjBag& SampleBag, Pix& p); // In the above two methods, the incoming value of p determines the
    // first object to include, while the final value reports the top
    // object not included in the sample. (p==NP) when and only when
    // the sample exhausts the wtdObjSet.

    virtual Bool isEmpty() {return(first() == NP);};
    virtual Bool isElt(fullID& ID, weightType& Wt); // If TRUE, set Wt.

    virtual int size(); // The total number of elements;
    virtual int numRemaining(Pix p); // The number of elements left after p.
    virtual int exactSize() = 0; // NOTE: the functions size() and
    virtual int approxSize(); // numRemaining() above are being
    virtual int maxSize(); // replaced by this more flexible
    // trio.

    virtual int exactNumRemaining(Pix p); // The exact*() methods are
    virtual int approxNumRemaining(Pix p); // often expensive; the others
    virtual int maxNumRemaining(Pix p); // are cheaper where possible.

    virtual char* toStr(); // Mostly for debugging: translate the entire
    virtual char* profile(); // set, or just a quick glance, into human-
                        // readable format.

Fig. WOSET_DEF: Class definition for C++ implementation of a weighted object set.

2.2 Basic Weighted Object Sets
[[ Empty wtdObjSets, Singleton wtdObjSets ]]
[[ Flat wtdObjSets. ]]
[[ Posting lists and posting sets. ]]
[[ Application: term inversion sets & Zipf’s law ]]

2.3. Searchers

Given the context of a weighted object-link model, we can give a rather precise definition of a
searcher (retrieval system; search engine). A searcher is (a computational object that provides) a
function from [patterns] in the [representation] into weighted sets, usually weighted sets of information objects that purport to fill some information need.

The simplest sort of searcher is a matching function on objects in a particular class. A class searcher takes an [information] specification and returns a weighted subset of objects in the class that match that description according to some computable measure of similarity. For both formal and computational simplicity, we will assume that searchers always return these sets in descending weight order — or more properly, non-increasing weight order — according to how well they match. When programming in an object-oriented environment, we regard such a function as a method of that class; in any implemented digital library, evaluating such a function includes making use of some [data set/database/collection] of the members of the class.

How does this work in a class system? Each object in the library/information system is assigned a particular class manager that is responsible for storing and identifying it. The class managers are set up to partition the complete universe represented in the system. In a flat class hierarchy, where the universe is simply partitioned into non-overlapping classes, this is simply done by assigning one manager to each proper class. In a rich class hierarchy with sub- and superclasses, decisions must be made on where to store each object. In MARIAN, for instance, the class individual serves as the superclass of person, corporation, and conference. The hasAuthor link connects works with individuals, but in point of fact the individual class manager is just a distribution point that directs class methods to the three subclasses and collects their results into a single class. All object storage and identification actually happens in one of the three subclasses.

Any class manager may implement matching methods conformant to the intended semantics of the class. Unless overridden, matching methods may also be inherited as usual. In MARIAN, for instance, the English root class is a descendent of the characterstring class. The characterstring class does not support approximate matching, but only a character-for-character exact match. [[Since identity in characterstring is a strict Boolean function, any match set returned by the characterstring searcher is either empty or a flat singleton set where the sole element has weight topWt. ]] English root, by contrast, supports a morphological match that is a true approximate match. Presented with a character string, the morphological searcher returns a weighted object set of all the roots in the collection that might be transformed to the string by common English morphological rules. The weights on these objects depend on how many and what types of transformations are required to produce the match. This function is central to parsing both text documents and user queries.

Link and weighted link classes have their own associated searchers. [[These searchers could also be specialized by classes and undergo inheritance. Thus far in our development of information and digital library systems at the Virginia Tech DLRL, however, we have not yet discovered a need to do so. ]] In the case of an unweighted link class, the class match method consists of accepting a weighted object set of nodes for either the source or sink of the link and producing the set of weighted objects at the other end of links in the collection.

The basic algorithm for this search is to expand the nodes of the key weighted object set, one at a time in nonincreasing order as they are given, to find all the target objects currently connected to it by links in the class, and to output those using the weight of the incoming node, optionally scaled by some class-determined factor. The only trick to this is that a single node in the target class may be linked to several nodes in the key set. Since we require that the output of any searcher function be a set, we must remove duplicates. The default method for this is to output each object the first time it is found and to maintain a table so that further references to the object may be filtered out. This has the effect of assigning each node its maximum weight among all alternatives. This is a common semantic rule — we submit, for instance, that it is the correct rule to use for hasAuthor and hasSubject links — but clearly not the only one possible.
Matching in a weighted link class has equally simple semantics, but requires a more complicated implementation. In this case as before the searcher outputs a set consisting of all the objects that are linked to any object in the key set. But here the weights in the result set depend on some function of the key node weight and the link weight (the default is simple multiplication). Given a range of node weights and a range of link weights, it is possible that the resulting objects will not be discovered in nonincreasing weight order. This has two computational effects: the input set must be explored to a greater depth (conceivably to the bottom if the weight ranges are sufficiently large) and more information must be maintained about the target nodes already seen. This in turn implies that a weighted link searcher is necessarily more computationally expensive than an unweighted link searcher.

It would also be possible in both weighted and unweighted link classes to provide searchers that boost the weight of a target node if it is linked to more than one key node. In the general case, one could use any arbitrary function to calculate resultant weights. This would involve additional computational complexity in link searchers. In practice, we have found that this complexity is only called for in more complex sorts of searches that can be better understood as intersection and union operations applied to the outcome of pattern searches.

### 2.4 Weighted Object Set Operations

Weighted object sets support the same operations as do classical sets: intersection, union, difference, cross product, and so forth. The intersection of two wtdObjSets, for instance, is defined as the weighted object set made up of all the members that belong to both sets; the union, the wtdObjSet made up of all the members of either. The problem with defining these operations comes in determining the weight function for the resulting set. Suppose we have two sets, each of which contain object \( a \), the first at weight \( w \) and the second at weight \( v \). Then both the intersection and union of the two sets contain object \( a \), but at what weight? \( w \), \( v \), or some other weight entirely?

[[ The weight-combining function is independent of the operator, although certain functions will prove more appropriate to certain operators ... ]] Some functions make better sense — have better semantics — in some operations; others in others. We have focussed on a few combinations that we believe to be fruitful. In the end, any functions and operations must be evaluated in context outside of information retrieval for their validity and utility. [[ formal analysis; user studies.]]

[[ maximum ]]

[[ summative functions:]]
- average & weighted average
- vector cosine for some direct product
- other vector space similarity functions: Croft, Harmon
- sigmoid
- relative entropy ]]

[[ quorum functions ]]

The cross product operator implies the existence of tuples, and ultimately relations and functions within the category of weighted objects. These important representation tools are finding a place in document retrieval in the context of hypertext.