This paper will appear in the Proceedings of the ACM SIGIR 1988 International Conference on Research and Development in Information Retrieval.

QUERY PROCESSING IN A HETEROGENEOUS RETRIEVAL NETWORK

Patricia Simpson

CS-TR-158-88

May 1988

Revised July 1988

QUERY PROCESSING IN A HETEROGENEOUS RETRIEVAL NETWORK

Patricia Simpson

Department of Computer Science
Princeton University
Princeton, NJ 08540

Abstract

The concept of a large-scale information retrieval network incorporating heterogeneous retrieval systems and users is introduced, and the necessary components for enabling term-based searching of any database by untrained end-users are outlined. We define a normal form for expression of queries, show that such queries can be automatically produced, if necessary, from a natural-language request for information, and give algorithms for translating such queries, with little or no loss of expressiveness, into equivalent queries on both Boolean and term-vector type retrieval systems. We conclude with a proposal for extending this approach to arbitrary database models.

May 19, 1988

QUERY PROCESSING IN A HETEROGENEOUS RETRIEVAL NETWORK

Patricia Simpson

Department of Computer Science
Princeton University
Princeton, NJ 08540

1. Overview

Present-day retrieval systems are larger, more numerous, and more sophisticated than ever before, yet usage of publicly available databases remains disconcertingly low. The problem of making computer-based information accessible to the population at large has a four-component solution: a reliable digital communication network connecting information providers to homes, businesses, and libraries; inexpensive personal computer systems (PCs) providing individuals with personal information storage and processing capabilities; easy-to-use query interfaces, customizable to each user, through which queries can be expressed without knowledge of online searching techniques or of the query languages of specific IR systems; and algorithms for translating queries submitted via such an interface into equivalent queries processable by any retrieval system attached to the network.

With the advent of ISDN the necessary public telecommunication networks are fast becoming reality. Such a network can be expected to provide the services of the first four layers of the ISO/OSI protocol, specifically message routing and reliable transmission of messages from node to node, and provide a base upon which an enormous variety of information-exchange applications may be implemented. We can thus envision a large-scale information network populated by both information providers and information seekers numbering, potentially, in the millions. Providers may include commercial information services, businesses, civic organizations, government agencies, professional societies, libraries, and individuals.

The second component is also in hand; within very few years we can expect PCs, in their role as communication and storage devices, to become as ubiquitous as telephones — provided, of course, that their ease of use can be made to approach that of telephones. With

This research was supported by the New Jersey Governor's Commission on Science and Technology Award No. 85-990660-6, and by grants from IBM and SRI's Sarnoff Laboratory.

PCs within the reach of the average person, it is reasonable to anticipate that individuals increasingly will store their important information in computer files and databases, as most businesses do now. It will also be desirable to exchange information with others, and to add to one's personal database items received directly from other machines.

This scenario has clear implications for the hundreds of public and commercial information services now in operation. The potential users exist; what is lacking is a truly ''friendly'' interface between them and information databases, rather than the current hodgepodge of query languages, thesauri, menus, manuals, and expensive subscriptions. These factors, intended to facilitate information access, often have the opposite effect of discouraging access — particularly in the case of the average person whose occasional information needs do not warrant the effort and expense of using present-day retrieval systems. A well-designed information network should enable users to query in a natural, uniform way any of the information providers attached to the net. This is not only a desirable enhancement of the present operation of remotely accessed information services, but an environment in which new services may easily arise. Even very small or specialized databases will find a large population of users ready to access them. An important design goal, therefore, is the provision of a single *unified* interface to the many information sources, but *customized* to each user. In particular, naive (untrained) users should be able to enter queries without having to learn the file structure, query language, or any other idiosyncrasies of a particular retrieval system, and to receive documents relevant to those queries without necessarily knowing how, or even by whom, they were answered.

While attempts have been made to define a single interface to multiple retrieval systems [MARCUS81, MOULINOUX83], the interface defined is fixed and not customizable to the requirements of individual users. On the other hand, a method for partially customizing interfaces for users of varying experience levels has been proposed [TAGUE87], but this centralized design, wherein a single system maintains records of usage patterns and preferences of its users, would not scale well to a large number of users.

We argue that it is possible to have it both ways; fully customized interfaces can be implemented on each user's PC, decoupling query entry (handled by the PC) from query processing (performed by the information service, or ''host''). In fact, a user's processor should be able to accept a query, locate hosts capable of answering it, query those hosts, and collect responses, integrating them into its own database. To achieve this separation of function we propose a simple common language for communication of queries. We will call this language, motivated in Section 2 and presented in detail in Section 3, *query normal form* (QNF).

There are two basic requirements to be made of QNF. First, it must be possible for PC software to automatically generate QNF queries from input provided by a naive user. In Section 4 we discuss how this can be done in a straightforward manner, even if the input is natural language prose. Second, QNF queries must be automatically translatable into the query language of any information retrieval system. Each host must take responsibility for performing this translation for itself. In Sections 5 and 6 we give algorithms for processing QNF queries on Boolean-model retrieval systems and term-vector systems respectively. In both cases a ranked collection of documents is produced in response to the query and transmitted to the user.

In Section 7 we consider possible extensions to query normal form which could enable it to be used with a wider variety of retrieval and database systems.

2. Querying in a Heterogeneous Environment

In designing an application for a network of national and perhaps international proportions, there are two characteristics that must be given serious consideration: autonomy and heterogeneity of participants. *Autonomy* means freedom from any sort of control by other nodes, and freedom to define as much as possible one's own view of the network and means of interaction with it. This implies the need for completely decentralized control and processing. A node must never be required to cooperate with other nodes. Standards should be minimal, and universal enough that they enhance, rather than inhibit, a node's ability to participate in the net. *Heterogeneity* of participants must be expected and accommodated. This encompasses hardware, software, storage systems, retrieval languages, and the needs and interests of users themselves. It is certainly unreasonable to expect much uniformity among computer systems in the near future. By concentrating on integrating heterogeneous systems rather than imposing uniformity, several benefits are obtained: present systems are already usable, new equipment and software are easily introduced, diversity of information services is encouraged, and nodes are free to upgrade their capabilities at any time without disrupting the operation of the network. Simplicity of design and operation is key to maximizing network participation. Ideally any node, even a small home computer, should be capable of functioning as both a *querier* (information seeker) and a *source* (information provider).

As a result of the stated goals of preserving autonomy and heterogeneity while providing a single interface to diverse databases, we propose a noninteractive, one-question/one-answer protocol for node-to-node communication. We will not require that any node support interaction with another, since that would necessitate each party keeping track of the state of the interaction until the session ends (violating the autonomy criterion) and would make it difficult to insulate the user from the host's own query language (violating the unified interface criterion). Therefore the language in which queries are expressed must be rich enough to express a complete information need, or question, in a single query.

While it is desirable to minimize the imposition of standards over the network so as not to restrict its inherent flexibility and diversity, it is necessary that hosts understand the queries submitted to them and that users understand the responses. Since responses are assumed to be human-readable documents, their representation as a string of natural-language text seems sufficient. Queries, on the other hand, must be machine-processable and so must be carefully designed. In a network of very many nodes it is clearly impractical to require direct translation between the query languages of every pair of processors; a practical approach is to impose some common query language, which we will call query normal form (QNF).

This query language, to be maximally expressive, must be based in natural language. It should be easily generatable by users and their processors. For the benefit of very naive users, it is important that automatic procedures exist to convert a natural-language prose request by such a user into a correct, equivalent QNF query. Furthermore, QNF queries must be processable by any host system on the network. Yet it would be undesirable to require hosts to make modifications to their file structures or retrieval operations in order to process QNF queries; both autonomy and heterogeneity would suffer. Instead, we ask only that each source perform some preprocessing to translate QNF queries into its own query language, then execute the resulting queries normally. Some postprocessing of the results may also be required. We will first present a normal form, then show that such queries can indeed be both generated automatically and translated to either term vector format or Boolean format for processing.

3. Query Normal Form

A QNF query is of the form <T,N,W> where T = $\{t_1/w_1 , . . . , t_n/w_n\}$ is a set of weighted terms, N is a positive integer indicating the maximum number of documents desired, and W is the minimum acceptable weight of a document, where a *document weight* is taken to be the sum of the weights of the query terms under which the document is indexed. Weights are rational numbers in the interval (0,1]. The querier uses them to indicate the relative importance of the terms to the query. A weight of 1 indicates a *required* term. Equally weighted terms have a special interpretation; they are taken to be *synonyms*. The constraints N and W combine with the term weights to give the user fairly fine-grained control over the ordering and quantity of the retrieved documents that will be returned. This is important because the characteristics of sources will vary widely — some may have only general information on a topic while others treat it in great depth; some may have thousands of relevant documents. By incorporating control information (N, W, and weights) into the query itself, some of the benefits of interactive access can be attained in a noninteractive environment. A QNF query includes an upper bound on quantity, a lower bound on quality, and a means of *ranking* responses so that the most relevant ones are returned first, even by sources (such as Boolean retrieval systems) that do not normally rank query responses.

Let us look at an example. The query

Q1: < {home/0.7 , water/1.0 , filter/0.9 , pure/0.9} , 20 , 1.9 >

expresses an interest in home water purification. No more than 20 documents are wanted, and only those indexed by one of the following minterms (an overbar denotes complementation):

home AND water AND filter AND pure
home AND water AND filter AND $\overline{\text{pure}}$
home AND water AND $\overline{\text{filter}}$ AND pure
$\overline{\text{home}}$ AND water AND filter AND pure
$\overline{\text{home}}$ AND water AND filter AND $\overline{\text{pure}}$
$\overline{\text{home}}$ AND water AND $\overline{\text{filter}}$ AND pure

There is no *single* Boolean query which incorporates all of the information in this QNF query. There is, however, a procedure for generating an equivalent *sequence* of Boolean queries, which will be presented in Section 5. The best one-line approximation to *Q1* is the pseudo-Boolean

B1: water AND (filter OR pure) ANDMAYBE home

''Water'' is a required term, ''filter'' and ''pure'' are ORed because they are synonymous, and ''home'' is optional. The construct ANDMAYBE indicates that the presence of ''home'' as an index term of retrieved documents is desirable, but not necessary; the practical interpretation of this is that documents containing ''home'' should be ranked above those lacking it.

As a second example, consider the query

Q2: < {home/0.7 , water/1.0 , filter/0.9 , pure/0.9} , 20 , 2.0 >

By increasing W to 2.0 we have composed a more selective query, equivalent to the union of:

home AND water AND filter AND pure
home AND water AND filter AND $\overline{\text{pure}}$
home AND water AND $\overline{\text{filter}}$ AND pure

or more succinctly:

B2: water AND (filter OR pure) AND home

''Home'' is no longer optional if the minimum weight specification is to be met; ''$\overline{\text{home}}$ AND water AND filter AND pure'' does not satisfy the W constraint because ''filter'' and ''pure'' are designated as synonyms — both terms together have the same weight (0.9) as either one alone. We might call this the *true synonym* interpretation of equal weights. An alternate interpretation is as *heavy synonyms*, where a little extra weight is given to minterms incorporating more than one term of a synonym group, on the assumption that they are slightly more relevant to the query than if only one were present. In this case m synonymous terms with weight w appearing uncomplemented in a minterm contribute $w + (m-1) \cdot w \cdot \varepsilon$ to the weight of that minterm, where ε is small. For example the six minterms given for query *Q1* were shown in ranked order; their values are, respectively, $2.6 + (0.9\,\varepsilon)$, 2.6, 2.6,

$1.9 + (0.9\,\varepsilon)$, 1.9, and 1.9. The host has the option of choosing which interpretation to implement.

As a final example to illustrate the meaning of term weights with regard to ranking, consider the query

Q3: < {home/0.5 , water/0.9 , filter/0.7} , 20 , 1.4 >

The applicable documents are those specified by:

home AND water AND filter	(total weight = 2.1)
$\overline{\text{home}}$ AND water AND filter	(" " = 1.6)
home AND water AND $\overline{\text{filter}}$	(" " = 1.4)

The single Boolean query

B3: water AND (home OR filter)

while logically equivalent to the union of the minterms given, is incapable of representing the user's preference for ''water , filter'' over ''home , water'' or for ''home , water , filter'' over either.

4. The User Query Interface

As already mentioned, the query entry procedure may be customized by each user to meet his own needs and preferences. We must guarantee, however, that *some* interface can be designed which will enable naive users to enter queries which are syntactically correct and which express their information need adequately.

It has been established that automatic indexing techniques applied to document texts can generate satisfactory document descriptors. By the same mechanisms, which are simple enough to implement on PC systems, query descriptors can be generated from natural language queries. Such a procedure, given in [SALTON83], consists of these steps:

1. decompose the NL text into a list of discrete words
2. delete low-content stopwords
3. remove suffixes to reduce words to word stems
4. eliminate duplicates, assigning higher weights to terms occurring more than once.

This process results in a single set of terms, possibly with weights based on frequency of occurrence in the NL query. The user can be prompted for a value for N, or some small default can be used. W can perhaps default to the lowest assigned term weight to effect high recall, or to the sum of most or all term weights to effect high precision. Natural language query interfaces to specific retrieval systems have already been implemented with success [BERNSTEIN84, DOSZKOCS83]; they can be implemented in this environment as well.

Of course, users willing to put more effort into creating queries can easily generate quite specific and expressive requests. One can readily envision an easy-to-use graphic interface permitting entry of words, adjustment of their weights, spelling correction, and so forth. The exact choice of weights is not particularly important; it is the relative weights of terms and the relationships of their sums to the W value that determine a query's meaning. Graphic aids to setting and understanding these relationships can also be devised.

At this point we should mention briefly a mechanism for choosing the information sources to which a given query is to be sent. Given a very large decentralized information network of changing composition, a complete, consistent, universally accessible index to its contents would be impossible to construct and maintain. Any index used will be a distributed, dynamic, and often inconsistent one; each node should store locally as much of this index as it wants and/or needs, and be able to query remote portions of the index when the local portion is insufficient or out-of-date. We call each node's portion of the index its *directory*. The directory is a list of node addresses together with summaries of the information each can provide. These summaries, which we will call *node descriptors* (NDs), can be supplied by the source nodes themselves. Nodes may acquire new NDs for their directories by asking neighbors (known source nodes) for copies of their directories, by asking source nodes for their own NDs, and by receiving unsolicited ''announcements'' of NDs from information sources.

An ND is a weighted term set, similar in format to the T component of a QNF query. When a user composes a query his processor will search its directory for NDs which match the query, using some similarity measure. (Inner product, with the weights of missing terms taken to be zero, is a simple and accurate measure to use.) If a good match is found, the query can be sent to the appropriate source(s). If not, the user's processor can automatically build its directory by requesting copies of neighbors' directories. Thus the pool of neighbors increases, and the process repeats until a promising source is found.

A consequence of heterogeneity is the possibility of incompatible vocabularies among user and host systems. Liberal use of synonym terms in a query can help to make the query understandable to a wide range of hosts, or a querier can use a synonym dictionary during the query/ND matching process. A host, likewise, may maintain a dictionary or thesaurus with which it can map query terms into index terms acceptable to its retrieval system. In general, however, a high degree of matching with index terms can be expected in queries simply because the source's ND contains only valid index terms, and it is (presumably) a close match between the QNF query and the ND that causes the query to be submitted in the first place. A small number of unrecognized terms may be dropped by the host before processing, but the retrieval result will entail lower precision and recall than intended. If too many mismatches occur, the host may choose not to process the query and to send instead to the querier a current copy of its ND, on the assumption that the querier's copy is incorrect or outdated.

Depending on the capabilities of a particular querier and host, the querying and translation processes may be smooth or awkward. All adjustments to queries during the process may result in a loss of expressiveness, and this double-translation technique (NL to QNF to host language) can be expected to produce less exact queries in general than a user trained on the host system could produce. However, it is a flexible design which permits each participating node to contribute *as best it can* to the translation. Some nodes will be more "intelligent" than others, but any node with minimal PC software can ask a question and receive some relevant response. Modular design and a simple communication protocol can open up database access to millions of untrained potential users.

5. Query Processing on a Boolean Retrieval System

Any QNF query can be translated into a sequence of Boolean queries in such a way that weights are interpreted properly and the N and W constraints are met. The response consists of a *list* of *sets* of documents such that all members of a set have equal document weights, no document appears more than once, and every member of a given set is returned before any members of a lower-weighted set. We now describe procedures for translating from QNF to Boolean queries.

5.1. All-Minterm Algorithm

The most general approach is to generate all of the possible minterms, exclude those in which one or more required terms (weight = 1.0) is complemented, rank those remaining by decreasing weight, and execute the Boolean query corresponding to each in turn until either the maximum number of documents N has been retrieved or the value of the next query is below the threshold W. This procedure is simple and effective. The full degree of document ranking requested via the QNF query can be provided by a Boolean retrieval system.

The drawback to this approach is that it is of exponential time complexity. A query of n terms yields 2^n minterms, each of which (except the fully complemented one) describes a set of documents at least partially relevant to the query. In the worst case, then, the host system must generate 2^n-1 distinct sets and transmit them in ranked order to the querier.

For this reason we present an alternative algorithm requiring the production of at most $2n-1$ sets for transmission. It is also simple to execute, and its linear time complexity should make it more palatable to those who might implement QNF query processing on a Boolean retrieval system. The price paid is the precision with which the ranking is done; it is no longer guaranteed that the weight of every document returned is at least as great as that of all documents which follow it, although an approximate ranking is maintained.

5.2. An Approximate-Ranking Algorithm

Step 1. Retrieve a core collection of document sets, one set per term weight, as follows.
(a) Retrieve the set of documents indexed under each separate term t_i, $1 \le i \le n$, to yield n document sets S_{t_i}, $1 \le i \le n$.

(b) For all S_{t_i} where $w_i = 1.0$ (documents indexed under required terms) execute the Boolean operation $\cap S_{t_i}$, yielding the intersection set $S_{1.0}$ of documents containing all the required terms. Discard the original sets S_{t_i}.

(c) For all $S_{t_i},...,S_{t_k}$ where $w_i = \ldots = w_k$, execute the Boolean operation $\cup S_{t_i},...,S_{t_k}$, yielding the union set S_{w_i} containing all documents indexed under at least one of the synonymous terms of weight w_i. Discard the original sets $S_{t_i},...,S_{t_k}$.

(d) Each of the remaining sets S_{t_i} pertains to a uniquely weighted term; rename them by their weights to S_{w_i}.

There are now k core document sets, each corresponding to one of the k distinct term weights from the query. The actual weights do not affect the remainder of this algorithm, so for simplicity's sake we will rename the sets to have integer subscripts ranging from 1 to k, in order by increasing term weight (S_1 represents the lowest-weighted term and S_k the highest-weighted).

Step 2. Generate and execute a sequence of $k-1$ queries producing the sets

$$\begin{array}{rcl} & & S_k \text{ AND } S_{k-1} \\ & & S_k \text{ AND } S_{k-1} \text{ AND } S_{k-2} \\ & & \vdots \\ R_1 & = & S_k \text{ AND } S_{k-1} \text{ AND} \ldots \text{AND } S_1 \end{array}$$

Set R_1, containing the highest-weighted documents, is the first *response set* and is transmitted to the querier. (At any point, if the cumulative size of the response sets exceeds N, the first N documents are sent and the search terminates).

Step 3. Execute $k-1$ queries producing the response sets

$$\begin{array}{rcl} R_2 & = & S_k \text{ AND } S_{k-1} \text{ AND} \ldots \text{AND } S_2 \text{ NOT } S_1 \\ R_3 & = & S_k \text{ AND } S_{k-1} \text{ AND} \ldots \text{NOT } S_2 \\ \vdots & & \vdots \\ R_k & = & S_k \text{ NOT } S_{k-1} \end{array}$$

At this point, all documents containing the term(s) of highest weight have been retrieved. If that highest weight is 1.0, then the search ends here.

Step 4. Execute the final $k-1$ queries yielding the response sets

$$\begin{array}{rcl} R_{k+1} & = & S_{k-1} \text{ NOT } S_k \\ R_{k+2} & = & S_{k-2} \text{ NOT } S_{k-1} \text{ NOT } S_k \\ \vdots & & \vdots \\ R_{2k-1} & = & S_1 \text{ NOT } S_2 \text{ NOT} \ldots \text{NOT } S_k \end{array}$$

If all $2k-1$ sets are transmitted, the user receives all documents indexed under at least one of the query terms. If the query specifies a nontrivial value for W, however (greater than the lowest term weight), not all sets should be returned. Choice of the exact cutoff point involves a precision/recall tradeoff: if we guarantee that no documents of weight < W will be returned, we cannot guarantee that all documents of weight ≥ W will be, and vice versa. (The only exception to this occurs when the distinct query term weights all differ from each other by a factor of two or more. In this case a perfect ranking of sets *will* be produced, although it will still not be as fine-grained as that produced by the $O(2^n)$ algorithm). The choice of interpretation of W can be made by the host.

Although steps 2 - 4 appear to perform $O(k^2)$ Boolean operations, only $O(k)$ operations are actually required if queries make use of the results of previous queries. As an example, suppose the following query, concerning outdoor activities in Canada, were received:

< {camp/1.0 , boat/0.7 , cave/0.6 , spelunk/0.6 , Canada/1.0 , mountain/0.4} , 100 , 2.5 >

The following Boolean queries could be executed. Note that sets are numbered as they are created, as in a typical interactive retrieval session; the numbers thereafter may be used to identify the sets. To the right of each is shown the weight, or range of weights, of documents in the set, followed by the effective query. Response sets are starred.

(1)	mountain	0.4-3.7		
(2)	cave OR spelunk	0.6-3.7		
(3)	boat	0.7-3.7		
(4)	camp AND Canada	2.0-3.7		
(5)	3 AND 4	2.7-3.7		$(\text{camp}\cap\text{Canada}) \cap \text{boat}$
(6)	2 AND 5	3.3-3.7		$(\text{camp}\cap\text{Canada}) \cap \text{boat} \cap (\text{cave}\cup\text{spelunk})$
(7)	1 AND 6	3.7	*	$(\text{camp}\cap\text{Canada}) \cap \text{boat} \cap (\text{cave}\cup\text{spelunk}) \cap \text{mountain}$
(8)	6 NOT 7	3.3	*	$(\text{camp}\cap\text{Canada}) \cap \text{boat} \cap (\text{cave}\cup\text{spelunk}) \cap \overline{\text{mountain}}$
(9)	5 NOT 6	2.7-3.1	*	$(\text{camp}\cap\text{Canada}) \cap \text{boat} \cap \overline{(\text{cave}\cup\text{spelunk})}$
(10)	4 NOT 5	2.0-3.0	*	$(\text{camp}\cap\text{Canada}) \cap \overline{\text{boat}}$
(11)	NOT 4	0.0-1.7		$\overline{(\text{camp}\cap\text{Canada})}$
(12)	11 NOT 3	0.0-1.0		$\overline{(\text{camp}\cap\text{Canada})} \cap \overline{\text{boat}}$
(13)	12 NOT 2	0.0-0.4		$\overline{(\text{camp}\cap\text{Canada})} \cap \overline{\text{boat}} \cap \overline{(\text{cave}\cup\text{spelunk})}$
(14)	3 AND 11	0.7-1.7	*	$\overline{(\text{camp}\cap\text{Canada})} \cap \text{boat}$
(15)	2 AND 12	0.6-1.0	*	$\overline{(\text{camp}\cap\text{Canada})} \cap \overline{\text{boat}} \cap (\text{cave}\cup\text{spelunk})$
(16)	1 AND 13	0.4	*	$\overline{(\text{camp}\cap\text{Canada})} \cap \overline{\text{boat}} \cap \overline{(\text{cave}\cup\text{spelunk})} \cap \text{mountain}$

Since (in this example) set 4 is required, execution would end after production of set 10 (the last set containing set 4). The W constraint is, however, not necessarily met by all documents in set 10. The host chooses whether to stop after set 9 or set 10.

This particular algorithm places emphasis on the highest-weighted term, retrieving all documents containing it before any not containing it. Thus it should perform well on queries containing required terms, and in general when the high-weighted terms are of much greater importance to the querier than the low-weighted ones. Other algorithms can be devised; there is, for example, an $O(n^2)$ algorithm which provides a finer ranking than the linear algorithm of the response sets not containing the highest-weighted term.

6. Query Processing on a Term-Vector Retrieval System

Since QNF is based on a weighted term set, processing by a retrieval system which represents documents as vectors of index term weights and expects weighted term-vector queries as well should be fairly straightforward. In fact, if query/document similarity is computed as the inner product of the two vectors (as will be assumed throughout this section), index weights are binary, and no synonyms or required terms are specified in the query, the T component can be processed as is. Response documents are then transmitted in ranked order, using N and W to set a cutoff point.

For the general case, where synonyms and/or required terms are present, some pre- and postprocessing must be done to combine the weights properly. In addition, if index weights may be nonbinary (real numbers in the interval [0,1]), W must be reinterpreted. We will treat binary and nonbinary index weights separately, beginning with the simpler (binary) case.

6.1. Binary Document Term Vectors

A term-vector retrieval system is a collection of r documents, $d_1, d_2, \cdots d_r$, indexed by s terms, $t_1, t_2, \cdots t_s$. The indexed collection is represented by an $r \times s$ matrix D in which the entry D_{ij} is 1 if document d_i pertains to term t_j and 0 if it does not. A query Q is an $s \times 1$ matrix whose entry Q_j is 1 if the query pertains to term t_j and 0 if it does not. (Our algorithm performs preprocessing to convert the QNF query into several binary query vectors. It therefore can be used by a host that accepts either binary or nonbinary queries.) The similarity of each document to a particular query Q is computed by multiplying $D \cdot Q$ to produce an $r \times 1$ matrix of similarity values.

Step 1. Convert the QNF query's T component into a vector of appropriate length by ordering its term weights to conform to the ordering of terms in D, and assigning weights of 0 to those index terms not in T. (For clarity in the examples, we will show only the nonzero-weighted query terms and the portion of D that pertains to them.) Let us use as an example the query

Q4 = < {Robert/1.0 , Frost/1.0 , style/0.8 , poem/0.3 , verse/0.3 , rhyme/0.3} , N , W >

and the document matrix

$$D = \begin{array}{c} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ \end{array} \left[\begin{array}{cccccc} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{array} \right]$$
$$\begin{array}{cccccc} \text{poem} & \text{rhyme} & \text{verse} & \text{style} & \text{Frost} & \text{Robert} \end{array}$$

The converted query is the vector $[0.3\ \ 0.3\ \ 0.3\ \ 0.8\ \ 1.0\ \ 1.0]$.

Step 2. We will define a *selector* query S for one or more terms t_j to be a binary $s \times 1$ vector whose elements S_j are 1 for each t_j to be selected, and 0 otherwise. Compose k selector queries, one for each distinct query weight in the original query. Thus for *Q4* three selectors are generated:

$$S^{0.3} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad S^{0.8} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad S^{1.0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Step 3. Execute each S^i, $0 < i \leq 1$, as a query, producing similarity vectors $V^i \leftarrow D \cdot S^i$:

$$V^{0.3} = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 1 \\ 2 \end{bmatrix} \qquad V^{0.8} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad V^{1.0} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 0 \end{bmatrix}$$

Step 4. Adjust each V^i, $0 < i < 1$ (not $V^{1.0}$), to account for the query weights. Either true or heavy synonyms may be implemented. For true synonyms:

$$\text{for all } i < 1, \quad \text{for all } j,\ 1 \leq j \leq r, \qquad V_j^{i\,\prime} \leftarrow i \cdot (V_j^i > 0)$$

(Logical expressions, such as $(V_j^i > 0)$, evaluate to 1 if true and 0 if false.) For heavy synonyms:

$$\text{for all } i < 1, \quad \text{for all } j,\ 1 \leq j \leq r, \qquad V_j^{i\,\prime} \leftarrow i \cdot (1 + (V_j^i - 1) \cdot \varepsilon) \cdot (V_j^i > 0)$$

In either case set $V^{1.0\,\prime} \leftarrow V^{1.0}$ if $V^{1.0}$ exists. In our example, assuming heavy synonyms, this step produces vectors:

$$V^{0.3\,\prime} = \begin{bmatrix} 0.3{+}0.3\varepsilon \\ 0.3{+}0.6\varepsilon \\ 0 \\ 0.3 \\ 0.3{+}0.3\varepsilon \end{bmatrix} \qquad V^{0.8\,\prime} = \begin{bmatrix} 0.8 \\ 0.8 \\ 0 \\ 0 \\ 0.8 \end{bmatrix} \qquad V^{1.0\,\prime} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 0 \end{bmatrix}$$

Step 5. Define *req* to be the number of required terms in the original query: $req \leftarrow [\,1\ 1\ 1\ 1\ 1\ 1\,] \cdot S^{1.0}$ if $S^{1.0}$ exists; $req \leftarrow 0$ otherwise. Then generate the final response vector V^R as follows:

$$\text{for all } j,\ 1 \leq j \leq r, \qquad V_j^R \leftarrow \left[\sum_{i=1}^{k} V_j^{w_i\,\prime} \right] \cdot (V_j^{1.0} = req)$$

For our example,

$$V^R = \begin{bmatrix} 0 \\ 3.1{+}0.6\varepsilon \\ 2.0 \\ 2.3 \\ 0 \end{bmatrix}$$

Step 6. V^R contains the desired document rankings, just as it would if an ordinary vector query had been processed. Synonyms, required terms, and weights have all been taken into account. Now transmit the N (or fewer) highest-ranked documents whose weights are W or greater to the querier in ranked order.

6.2. Nonbinary Document Term Vectors

Although the presence of nonbinary weights in D makes query processing somewhat more complicated, an algorithm very similar to the one just presented can be used. Let us use as an example the document matrix

$$D = \begin{array}{c} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ \\ \end{array} \begin{array}{cccccc} 0.1 & 0.7 & 0.9 & 0.7 & 0.9 & 0.0 \\ 0.9 & 0.6 & 0.8 & 0.8 & 0.4 & 0.7 \\ 0.0 & 0.0 & 0.1 & 0.0 & 0.6 & 0.6 \\ 0.1 & 0.0 & 0.7 & 0.2 & 0.3 & 0.9 \\ 0.0 & 0.5 & 0.6 & 0.8 & 0.0 & 0.1 \\ \text{poem} & \text{rhyme} & \text{verse} & \text{style} & \text{Frost} & \text{Robert} \end{array}$$

Step 1. Convert the T component to a vector as before.

Step 2. Generate a selector query for each term in the original query. Where m query terms share a weight w, their selector queries are labeled $S^{w^1}, \ldots, S^{w^m}$:

$$S^{0.3^1} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad S^{0.3^2} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad S^{0.3^3} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad S^{0.8^1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad S^{1.0^1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad S^{1.0^2} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Step 3. For each S^i, $0<i\leq 1$, compute similarity vector $V^i \leftarrow D \cdot S^i$:

$$V^{0.3^1} = \begin{bmatrix} 0.1 \\ 0.9 \\ 0 \\ 0.1 \\ 0 \end{bmatrix} \quad V^{0.3^2} = \begin{bmatrix} 0.7 \\ 0.6 \\ 0 \\ 0 \\ 0.5 \end{bmatrix} \quad V^{0.3^3} = \begin{bmatrix} 0.9 \\ 0.8 \\ 0.1 \\ 0.7 \\ 0.6 \end{bmatrix} \quad V^{0.8^1} = \begin{bmatrix} 0.7 \\ 0.8 \\ 0 \\ 0.2 \\ 0.8 \end{bmatrix} \quad V^{1.0^1} = \begin{bmatrix} 0.9 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0 \end{bmatrix} \quad V^{1.0^2} = \begin{bmatrix} 0 \\ 0.7 \\ 0.6 \\ 0.9 \\ 0.1 \end{bmatrix}$$

Step 4. Adjust each V^i to account for query weights and for synonym relationships. The fuzzy Boolean OR operator, **max**, can be used to implement true synonyms:

$$\text{for all } i<1, \quad \text{for all } j,\ 1 \le j \le r, \qquad V_j^{i\,\prime} \leftarrow i \cdot \max_{x=1}^{m} V_j^{i^x}$$

or heavy synonyms may be implemented instead:

$$\text{for all } i<1, \quad \text{for all } j,\ 1 \le j \le r, \qquad V_j^{i\,\prime} \leftarrow i \cdot \left[\max_{x=1}^{m} V_j^{i^x} + \left[\sum_{x=1}^{m} V_j^{i^x} - \max_{x=1}^{m} V_j^{i^x} \right] \cdot \varepsilon \right]$$

Both formulas are generalizations of the binary-vector formulas given in the previous algorithm. Index weights of required terms can be combined with the fuzzy Boolean AND operator, **min**. Where *req* is the number of required query terms:

$$\text{for all } j,\ 1 \le j \le r, \qquad V_j^{1.0\,\prime} \leftarrow req \cdot \min_{x=1}^{m} V_j^{1.0^x}$$

In the present example, assuming heavy synonyms, these adjustments produce the vectors:

$$V^{0.3\,\prime} = \begin{bmatrix} 0.27{+}0.24\varepsilon \\ 0.27{+}0.42\varepsilon \\ 0.03 \\ 0.21{+}0.03\varepsilon \\ 0.18{+}0.15\varepsilon \end{bmatrix} \qquad V^{0.8\,\prime} = \begin{bmatrix} 0.56 \\ 0.64 \\ 0 \\ 0.16 \\ 0.64 \end{bmatrix} \qquad V^{1.0\,\prime} = \begin{bmatrix} 0 \\ 0.80 \\ 1.20 \\ 0.60 \\ 0 \end{bmatrix}$$

Step 5. Compute the final response vector V^R as follows:

$$\text{for all } j,\ 1 \le j \le r, \qquad V_j^R \leftarrow \left[\sum_{i=1}^{k} V_j^{w_i\,\prime} \right] \cdot (V_j^{1.0\,\prime} > 0)$$

For our example,

$$V^R = \begin{bmatrix} 0 \\ 1.71{+}0.42\varepsilon \\ 1.23 \\ 0.97{+}0.03\varepsilon \\ 0 \end{bmatrix}$$

Step 6. In Boolean retrieval systems and vector systems with binary index weights, the threshold parameter W has been viewed as the minimum acceptable sum of the weights of those query terms applicable to each document. When index weights are nonbinary, however, the value of a query term counted toward a document weight will be less than that term's query weight whenever its index weight is less than 1.0. To compensate for this, W can be adjusted as follows:

$$W' \leftarrow \left[\frac{W}{\sum_{i=1}^{k} w_i + (req-1)\cdot(req>0)} \right] \cdot \left[\sum_{i=1}^{k} (w_i)^2 + (req-1)\cdot(req>0) \right]$$

W is divided by the maximum possible document weight to yield the minimum acceptable *proportion* of that maximum weight. W′ is set to that proportion of the weight of a document whose index weights for all query terms are the same as their query weights. Thus, for the present example:

$$W' = \frac{W}{0.3+0.8+1.0+1.0} \cdot (0.3^2+0.8^2+1.0+1.0) = \frac{2.73}{3.1} W$$

so that a document whose vector is [0.3 0.3 0.3 0.8 1.0 1.0], having a document weight of 2.73, would exactly meet the threshold if W = 3.1. Only in a nonbinary weighted system do the *exact* relationships among query weights affect the meaning of the query, indicating acceptable tradeoffs among terms. For example, a query term weighted 0.8 is "worth" twice as much as one weighted 0.4; the query $< \{t_1/0.8\ ,\ t_2/0.4\}\ ,\ N\ ,\ W >$ matches document vectors [0.8 0.4], [0.9 0.2], and [0.5 1.0] equally (the document weight is .80).

7. Future Work

Query normal form is an extension of the term-vector format for queries. It is natural-language based, simple, and easily generated and interpreted. However, while more expressive than simple weighted term vectors, QNF does not incorporate the full expressiveness of Boolean combinations of terms. For example, (a AND b) OR (c AND d) cannot be expressed in a single QNF query. While a generalization of QNF to weighted terms combined by Boolean operators would yield a more complete query language, we have in effect traded away some expressiveness for simplicity. Naive users have difficulty formulating Boolean queries correctly, and automatic translation from natural-language prose to a Boolean combination of terms is a difficult problem requiring a degree of language understanding that cannot be expected of a home PC system.

Nevertheless, it may be worth considering extensions to QNF which would increase its expressive power, adding negation, for example, or allowing phrases as search terms. One possible strategy for adding such features involves attaching to a query information about relationships between terms in the query (e.g., phrase membership, synonymity, hierarchical inclusion). The attachments would be optional, and hosts will vary in their ability to interpret them, but at least a mechanism would exist for those nodes that wish to communicate in a more sophisticated manner. These possibilities are currently being investigated.

A logical next step is the development of techniques for translating QNF queries into the query languages of general purpose (non-IR) database systems. This is a more challenging endeavor, for several reasons. The variety of query languages is greater, and some are procedural, requiring generation of a small program to execute a single QNF query. QNF queries may contain any NL term, whereas database schemas are extremely terse, containing only names for files and fields, which may bear little resemblance to natural language words.

The data dictionary or some other auxiliary device must be used to map QNF terms into file and field names, and additional work is needed to associate terms specifying desired field values with the appropriate files and fields. There is no ready-made concept of a document as the unit of retrieval, and the information retrieved (e.g. a collection of stored values) is out of context. Values have no meaning unless they are associated with a description of the containing records and/or files, as well as of the meaning of the field (or attribute) which they instantiate. The questions of what values to return and how to put them in context in the response are difficult, but must be answered if arbitrary database systems are to participate in the network.

8. Conclusions

We have outlined a design for a large-scale information network composed of heterogeneous autonomous computer systems. Solutions were suggested for some of the basic problems in implementing information retrieval over such a network: customizing the query interface to individual users, discovering new information sources and choosing the proper source to query, generating queries from natural-language input, and, most importantly, translating queries for execution by diverse retrieval systems. Specific translation methods were presented for term-vector and Boolean IR systems, and some of the issues involved in translating to more general database models were discussed.

Acknowledgments

This research was supported by the New Jersey Governor's Commission on Science and Technology Award No. 85-990660-6, and by grants from IBM and SRI's Sarnoff Laboratory. It was supervised by Professor Rafael Alonso, whose support and encouragement are greatly appreciated.

References

[BERNSTEIN84]

Bernstein, Lionel M. and Williamson, Robert E. ''Testing of a Natural Language Retrieval System for a Full Text Knowledge Base'', *Journal of the American Society for Information Science*, vol. 35, no. 4 (July 1984), pp. 235-247.

[DOSZKOCS83]

Doszkocs, Tamas E. ''From Research to Application: The CITE Natural Language Information Retrieval System'', *Research and Development in Information Retrieval, Proceedings*, Berlin, May 1982, G. Salton and H.-J. Schneider, eds., pp. 251-262. Springer-Verlag, New York, 1983.

[MARCUS81]

Marcus, Richard S. and Reintjes, J. Francis. ''A Translating Computer Interface for End-User Operation of Heterogeneous Retrieval Systems. I. Design. II. Evaluations'', *Journal of the American Society for Information Science*, vol. 32, no. 4 (July 1981), pp. 287-317.

[MOULINOUX83]

Moulinoux, C., Faure, J. C. and Litwin, W. ''MESSIDOR: A Distributed Information Retrieval System'', *Research and Development in Information Retrieval, Proceedings*, Berlin, May 1982, G. Salton and H.-J. Schneider, eds., pp. 51-61. Springer-Verlag, New York, 1983.

[SALTON83]

Salton, Gerard and McGill, Michael J. *Introduction to Modern Information Retrieval*, McGraw-Hill Book Company, New York, 1983.

[TAGUE87]

Tague, Jean. ''Generating an Individualized User Interface: From Novice to Expert'', *Proceedings of the Tenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, June 1987, pp. 57-60.